



Universidade de Coimbra  
Faculdade de Ciências e Tecnologia  
Departamento de Engenharia Informática

## **Relatório de Estágio**

Ano lectivo 2007/2008

# **Ruby on Rails**

## Desenvolvimento de Aplicações Web

**Ricardo Horta e Vale Otero dos Santos**

**Mentes Virtuais Lda.**

Supervisão na UC: Prof. Carlos Vaz

Supervisão na Mentes Virtuais: Eng. Marcos Garcia

14 de Julho de 2008

# Resumo

---

O presente relatório descreve o trabalho efectuado durante o estágio de carácter profissional integrado no Mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra. Descreve-se assim todo o estudo e desenvolvimento dos resultados finais, assim como todo o contexto e actividades em que o estagiário esteve envolvido. Tenta-se assim manter o equilíbrio entre a metodologia utilizada, os processos envolvidos e os resultados obtidos, descrevendo os momentos que constituíram a experiência profissional em contexto empresarial, um importante componente na formação do aluno.

Numa primeira fase fez-se um estudo da *framework* Ruby on Rails, bem como a sua experimentação, pondo em prática os conhecimentos adquiridos. Foi também feito um comparativo com outras alternativas ao desenvolvimento de aplicações web, focando a facilidade e rapidez de desenvolvimento. Esse comparativo envolveu o desenvolvimento de uma pequena aplicação de teste em Ruby on Rails, Django, CakePHP, Zope+Plone e Java EE.

Com base num conhecimento mais sólido, o estagiário desenvolveu uma aplicação de anúncios - *RoomBlick*, onde puderam ser estudadas algumas alternativas a problemas concretos no desenvolvimento da aplicação, bem como as soluções mais adequadas. Salientam-se tarefas como a integração com OpenID, georreferenciação, internacionalização e localização de uma aplicação, implementação de uma pesquisa avançada eficiente e a integração com um *gateway* de SMS. Foram também abordados problemas intimamente relacionados com o desenvolvimento de uma aplicação web usando Ruby on Rails como os mecanismos de *caching* quando em ambientes de produção, a realização de testes e a usabilidade.

As soluções encontradas, bem como a própria clarificação de conceitos e a descrição da implementação de algumas soluções são de extremo interesse para a empresa em que o estágio foi realizado, adquirindo-se um *know-how* no desenvolvimento com a *framework* Ruby on Rails.

# Índice

---

|   |    |
|---|----|
| 1. Introdução .....                                   | 1  |
| 1.1. Contextualização .....                           | 1  |
| 1.2. Motivação e objectivos .....                     | 1  |
| 1.3. Instituição .....                                | 2  |
| 1.4. Ambiente de desenvolvimento .....                | 2  |
| 1.5. Método de trabalho na empresa .....              | 3  |
| 2. Planeamento .....                                  | 5  |
| 3. Ruby on Rails .....                                | 6  |
| 4. Estudo comparativo de <i>frameworks</i> .....      | 7  |
| 4.1. Introdução.....                                  | 7  |
| 4.2. Ruby on Rails.....                               | 8  |
| 4.3. Django .....                                     | 8  |
| 4.4. Zope+Plone .....                                 | 10 |
| 4.5. CakePHP .....                                    | 12 |
| 4.6. Java EE .....                                    | 13 |
| 4.7. Conclusões.....                                  | 15 |
| 5. Roomblick.....                                     | 18 |
| 5.1. Introdução.....                                  | 18 |
| 5.2. Riscos Iniciais .....                            | 19 |
| 5.3. Outras soluções.....                             | 19 |
| 5.4. Implementação .....                              | 19 |
| 5.4.1. Migração para Rails 2.0 e filosofia REST ..... | 20 |
| 6. Integração com tecnologias.....                    | 21 |
| 6.1. Introdução.....                                  | 21 |
| 6.2. OpenID.....                                      | 21 |
| 6.2.1. Implementação .....                            | 22 |
| 6.2.2. Conclusões .....                               | 23 |
| 6.3. Georreferenciação .....                          | 24 |
| 6.3.1. Pontos de interesse .....                      | 24 |

|   |           |
|---|-----------|
| 6.3.2. Apresentação e cálculo de distâncias.....        | 25        |
| 6.3.3. Pesquisa por morada.....                         | 26        |
| 6.3.4. Exportação de dados .....                        | 27        |
| 6.3.5. Conclusão.....                                   | 28        |
| <b>6.4. Internacionalização e localização .....</b>     | <b>28</b> |
| 6.4.1. Internacionalização e localização em Rails ..... | 29        |
| 6.4.2. Conclusão.....                                   | 34        |
| <b>6.5. Pesquisa .....</b>                              | <b>35</b> |
| <b>6.6. Integração com <i>gateway</i> SMS.....</b>      | <b>37</b> |
| 6.6.1. Troca de mensagens .....                         | 38        |
| 6.6.2. Segurança .....                                  | 39        |
| <b>7. Optimização, testes e usabilidade.....</b>        | <b>41</b> |
| <b>7.1. Introdução.....</b>                             | <b>41</b> |
| <b>7.2. Optimização e caching .....</b>                 | <b>41</b> |
| 7.2.1. Counter cache.....                               | 43        |
| 7.2.2. Action Cache e Page Cache .....                  | 45        |
| 7.2.3. Fragment cache .....                             | 46        |
| 7.2.4. Cache em Rails 2.1 .....                         | 48        |
| 7.2.5. Conclusão.....                                   | 50        |
| <b>7.3. Testes .....</b>                                | <b>51</b> |
| 7.3.1. Test Driven Development.....                     | 52        |
| 7.3.2. Behaviour Driven Development.....                | 52        |
| 7.3.3. Conclusão.....                                   | 54        |
| <b>7.4. Usabilidade .....</b>                           | <b>55</b> |
| 7.4.1. Testes de usabilidade .....                      | 56        |
| 7.4.2. Conclusão.....                                   | 58        |
| <b>8. Conclusões .....</b>                              | <b>59</b> |
| 8.1.1. Trabalho futuro .....                            | 60        |

# Índice de figuras

---

|  |    |
|--|----|
| Figura 1 - Exemplo de uma página de um projecto usando redmine.....                  | 3  |
| Figura 2 - Ciclo de desenvolvimento de um projecto.....                              | 4  |
| Figura 3 - Página de adição de um comentário no backoffice de Django.....            | 9  |
| Figura 4 - Aspecto geral da página de administração em Zope+Plone.....               | 11 |
| Figura 5 - Criação de um ficheiro de configuração para ligação à base de dados. .... | 12 |
| Figura 6 - Diferença de sintaxe entre o ficheiro de rotas em Django e RoR.....       | 15 |
| Figura 7 - Diferenças na definição do modelo de utilizador em CakePHP e RoR.....     | 16 |
| Figura 8 - Funcionamento da autenticação usando OpenID.....                          | 22 |
| Figura 9 - Alternativa de login com OpenID na aplicação RoomBlick. ....              | 24 |
| Figura 10 - Localização geográfica de um anúncio com API do Google Maps. ....        | 25 |
| Figura 11 - Ajuda visual para adição das coordenadas de um quarto.....               | 26 |
| Figura 12 - Informação geográfica de um quarto usando o Google Earth. ....           | 27 |
| Figura 13 - Vista da aplicação de teste dos plugins de internacionalização.....      | 31 |
| Figura 14 - Comparativo nas traduções de Gettext Localize e Simple Localization...   | 32 |
| Figura 15 - Janela de edição da tradução de uma aplicação no Poedit. ....            | 33 |
| Figura 16 - Tempos médios de render em diferentes plugins.....                       | 34 |
| Figura 17 - Página de pesquisa do RoomBlick .....                                    | 36 |
| Figura 18 - Funcionamento da plataforma PS2 .....                                    | 38 |
| Figura 19 - Ciclo de validação de um anúncio. ....                                   | 38 |
| Figura 20 - Vista de teste da aplicação RoomBlick.....                               | 42 |
| Figura 21 - Caso típico onde faz sentido aplicar counter cache. ....                 | 44 |
| Figura 22 - Tempo médio para página de utilizador com counter caching. ....          | 44 |
| Figura 23 - Total de queries para página de utilizador com counter caching.....      | 45 |
| Figura 25 - Tempo médio para pedidos com diversos mecanismos de caching.....         | 46 |

|   |    |
|---|----|
| Figura 25 - Página de detalhes de utilizador dividida em dois blocos distintos..... | 47 |
| Figura 26 - Total de queries para página de utilizador com todos os mecanismos .... | 47 |
| Figura 27 - Tempo para pedidos usando todos os mecanismos de caching.....           | 48 |
| Figura 28 - Tempo médio usando fragment caching com recurso a memcached.....        | 49 |
| Figura 29 - Resultado de testes unitários com RSpec .....                           | 53 |
| Figura 30 - Comodidades de um quarto em lista sequencial .....                      | 57 |
| Figura 31 - Comodidades, apresentando uma lista com todas as possíveis. ....        | 58 |

# Índice de tabelas

---

|  |    |
|--|----|
| Tabela 1 - Distribuição temporal das principais tarefas numa panorâmica geral..... | 5  |
| Tabela 2 - Quadro resumo das diferenças na implementação da aplicação de teste..   | 17 |
| Tabela 3 - Quadro comparativo de alguns plugins de internacionalização. ....       | 31 |
| Tabela 4 - Descrição dos parâmetros enviados por POST pela plataforma PS2.....     | 39 |
| Tabela 5 - Lista de alterações relacionadas com usabilidade .....                  | 55 |

# Glossário

---

| Termo       | Descrição   |
|-------------|---|
| AJAX        | Asynchronous Javascript and XML   |
| API         | Application Programming Interface   |
| Bottleneck  | Fenómeno onde a <i>performance</i> ou capacidade de um sistema fica limitado por um único componente                                      |
| Bug         | Erro ou falha de <i>software</i> que impede o seu correcto funcionamento  |
| CMS         | Content Management System   |
| CRUD        | Operações básicas sobre uma base de dados ( <i>create, read, update, delete</i> )   |
| CSV         | <i>Comma Separated Values</i> . Formato de ficheiro com valores separados por vírgulas.   |
| Deployment  | Disponibilização de uma aplicação para uso  |
| DOM         | Document Object Mapping   |
| Framework   | Estrutura de suporte ao desenvolvimento de um projecto de <i>software</i>   |
| Hash        | Conjunto de caracteres gerado a partir de dados existentes  |
| HTML        | HyperText Markup Language   |
| HTTP        | HyperText Transfer Protocol   |
| IDE         | <i>Integrated Development Environment</i>   |
| I/O         | <i>Input / Output</i>   |
| Java        | Linguagem de programação orientada a objectos   |
| Java EE     | Java Platform, Enterprise Edition   |
| JavaScript  | Linguagem de <i>scripting</i> que permite adicionar funcionalidades a páginas HTML  |
| Know-how    | Conhecimento adquirido acerca da forma de execução de uma tarefa  |
| MVC         | Design pattern Model-View-Controller  |
| Open Source | Conjunto de princípios e práticas que promovem o acesso livre ao design da aplicação permitindo a partilha de conhecimento sem restrições |
| ORM         | Object Relational Mapping   |
| Overhead    | Utilização indirecta de recursos, em excesso, no cumprimento de uma tarefa  |
| Perl        | Linguagem de programação dinâmica   |
| PHP         | Linguagem de programação concebida para produzir aplicações web dinâmicas   |



|             |   |
|-------------|---|
| Plugin      | <i>Software</i> que se integra com uma aplicação para determinada função                                  |
| Python      | Linguagem de programação de sintaxe minimalista   |
| Rails       | Termo frequentemente utilizado para a <i>framework</i> Ruby on Rails                                      |
| REST        | Representational State Transfer   |
| RoR         | Termo frequentemente utilizado para a <i>framework</i> Ruby on Rails                                      |
| RPC         | Remote Procedure Call   |
| Scripts     | <i>Software</i> que controla uma aplicação estando dependente desta                                       |
| SMS         | Short Messaging Service   |
| SOAP        | Simple Object Access Protocol   |
| SQL         | Structured Query Language   |
| SVN         | <i>Subversion</i> – sistema de controlo de versões  |
| Software    | Conjunto de instruções computacionais que desempenham determinada tarefa                                  |
| Standard    | Prática recomendada no processo de desenvolvimento  |
| TDD         | Test Driven Development   |
| Template    | Padrão de desenvolvimento   |
| URL         | <i>Uniform Resource Locator</i> – endereço de um recurso numa rede  |
| Web         | Termo frequentemente utilizado para World Wide Web  |
| Web 2.0     | Tendência actual no <i>web design</i> e desenvolvimento web, chamada a segunda geração das aplicações web |
| Web Service | Sistema de <i>software</i> desenhado para suportar interoperabilidade através de uma rede                 |
| XHTML       | eXtensible HyperText Markup Language  |
| XML         | eXtensible Markup Language  |

# 1. Introdução

---

## 1.1. Contextualização

A arquitectura base das aplicações web não é muito variável, e como tal é possível construir um suporte comum. Surgem assim as *frameworks* de desenvolvimento de aplicações web, construídas para facilitar a modularidade e a não repetição de código, reduzindo o tempo de desenvolvimento, consequentemente os custos de produção, aumentando a rentabilidade da empresa.

Ruby on Rails é uma das *frameworks* com mais entusiastas actualmente, pela sua simplicidade e agilidade, tendo sido adoptada pela Mentis Virtuais nos últimos anos e havendo já resultados bastante satisfatórios. No entanto ainda falta *know-how* em algumas tecnologias envolvidas ou das melhores práticas no desenvolvimento de aplicações, dado que a migração é recente e a própria *framework* tem apenas quatro anos, sofrendo constantes alterações desde então.

Sendo uma *framework* relativamente recente, as novidades à sua volta são frequentes, existindo assim uma nuvem de conhecimento disperso e muitas vezes pouco claro. Para se desenvolver aplicações em Ruby on Rails de forma sistemática e eficaz importa por isso estudar e decidir quais as melhores escolhas.

## 1.2. Motivação e objectivos

A empresa onde foi realizado o estágio migrou recentemente o processo de desenvolvimento para a utilização de Ruby on Rails, por ser aquela que pareceu mais indicada e ao mesmo tempo a que causou mais entusiasmo.

O objectivo deste estágio é realizar um estudo sobre a *framework* em questão, das suas potencialidades e tecnologias envolvidas, de modo a otimizar a sua utilização na empresa e criar bases sólidas de conhecimentos em Ruby on Rails com valor para os seus colaboradores. Ao mesmo tempo pretende-se também determinar objectivamente se a escolha efectuada foi adequada aos projectos aí desenvolvidos.

Para isso, desenvolveu-se paralelamente uma aplicação - *Roomblick*, cujo desenvolvimento tem valor para a empresa mas também tem como objectivo identificar quais as melhores soluções para determinadas funcionalidades que podem ser comuns a outras aplicações desenvolvidas utilizando a *framework* Ruby on Rails.

### 1.3. Instituição

A associação de empresas para uma rede de inovação em Aveiro - INOVA-RIA, constituída a 29 de Julho de 2003, tem por objectivo a consolidação de um *cluster* de telecomunicações que contribua para o desenvolvimento da região de Aveiro.

Do conjunto de empresas é possível destacar associados como a PT Inovação, a grande impulsionadora desta associação empresarial. Existem ainda outros associados não menos relevantes dos quais importa realçar a Mentis Virtuais, empresa onde foi efectuado o estágio curricular. A empresa existe desde Janeiro de 2003 e possui um vasto conhecimento na concepção de aplicações web e serviços móveis sustentado pela experiência dos seus colaboradores.

### 1.4. Ambiente de desenvolvimento

O ambiente de desenvolvimento usado neste projecto é constituído por várias ferramentas e, fundamentalmente, a *framework* Ruby on Rails. Todas as tecnologias envolvidas têm como característica comum serem *Open Source*, fomentando a sustentabilidade e independência do projecto em relação a tecnologias proprietárias.

As versões de Ruby on Rails utilizadas foram sendo gradualmente actualizadas ao longo do projecto desde a versão 1.2.3 até à versão 2.1. Assim, todas as ferramentas incluídas na *framework* também correspondem às versões enumeradas.

Para testes de *performance* recorreu-se à ferramenta *ApacheBench* incorporada no servidor web Apache, uma vez que a sua utilização foi feita para testar a *performance* de diferentes tipos de *caching* em aplicações a correr em *Phusion Passenger* - uma alternativa ao *deploy* em Ruby on Rails que funciona como um módulo de Apache.

## 1.5. Método de trabalho na empresa

O desenvolvimento de um produto na Mentis Virtuais, principalmente na área de aplicações web, não implica o desenvolvimento de documentação extensiva, uma vez que as equipas de trabalho são pequenas (duas a quatro pessoas). Existe um grande diálogo entre os elementos de um projecto, sendo encorajada a troca de ideias e conhecimento entre eles.

Enquanto o modelo clássico em cascata funciona bem para grandes aplicações web monolíticas, o desenvolvimento de uma aplicação desta natureza beneficia de uma aproximação iterativa e rápida. À medida que se desenvolve a aplicação, tentam evitar-se passos que levem a um ponto sem retorno. Todas as decisões tomadas devem ser rapidamente reversíveis se for tomado o caminho errado. Da mesma maneira, novas funcionalidades devem ser facilmente implementadas em pouco tempo, mesmo que perto da fase final (ou depois dessa fase). Quanto mais cedo houver uma aplicação funcional, mais cedo se encontra algum problema de *design* e menos tempo vai ser gasto para encontrar alternativas.

Na empresa é usado num sistema de gestão de projectos – Redmine [1] instalado num servidor interno onde se pode seguir o desenvolvimento de um projecto, bem como adicionar ou editar documentação já realizada, ficando acessível imediatamente, de modo simples a todos os elementos, podendo haver colaboração na sua especificação (Figura 1). Este sistema permite ainda alertar por email os participantes de um projecto de alterações efectuadas, criar um diagrama de Gantt e consultar um calendário de metas.

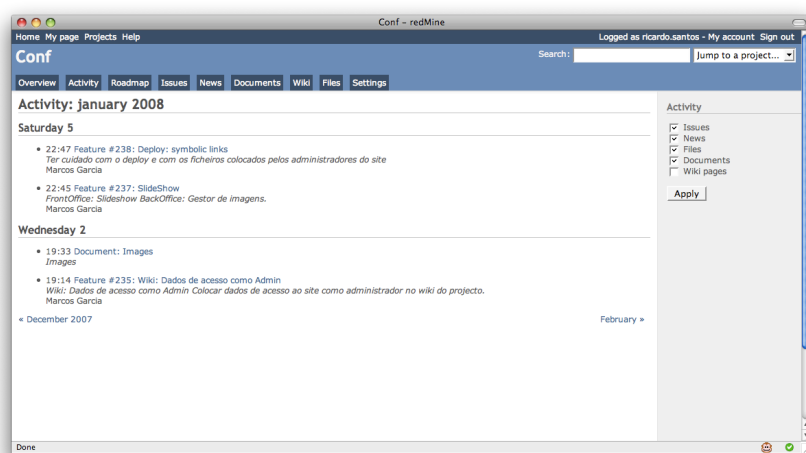
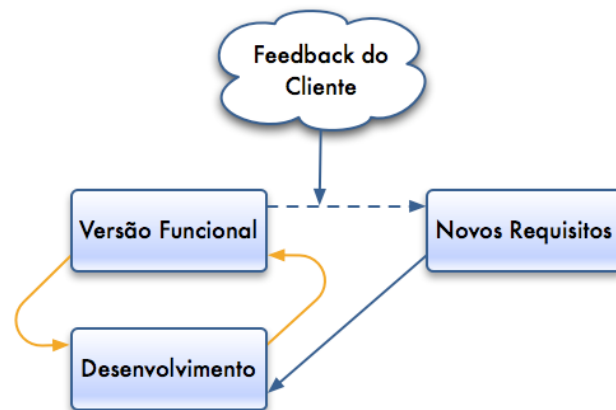


Figura 1 – Exemplo de uma página de um projecto usando o gestor de projectos redmine.

Além disso todos os projectos têm um repositório de controlo de versões, sendo usado um servidor SVN para o efeito. Desta forma anulam-se os problemas adjacentes ao desenvolvimento em paralelo por diferentes pessoas numa equipa.

A primeira versão funcional da aplicação é sempre posta em produção o mais rapidamente possível para que o cliente possa dar o seu parecer e normalmente surgem novos requisitos. À medida que as novas funcionalidades são implementadas a aplicação em produção é actualizada, obedecendo-se ao ciclo presente na Figura 2.



*Figura 2 – Ciclo de desenvolvimento de um projecto. A amarelo está o ciclo de desenvolvimento sem recurso ao cliente.*

## 2. Planeamento

---

O estágio foi desenvolvido em duas fases, havendo uma primeira fase, iniciada no dia 10 de Setembro de 2007, onde foi feita a ambientação à empresa, tecnologias e ferramentas bem como um primeiro levantamento de requisitos e o desenvolvimento de dois módulos simples para solidificação de conhecimentos. Esta fase correspondeu a um semestre lectivo com uma carga de 16 horas semanais com término no dia 21 de Janeiro de 2008. A segunda fase, com uma carga de 40 horas semanais começou no dia 10 de Fevereiro, findando dia 11 de Julho, com a elaboração deste relatório.

O plano inicialmente proposto para o desenvolvimento do projecto, sofreu algumas alterações com o intuito de o fazer corresponder à realidade e adaptá-lo a necessidades pontuais. Na Tabela 1 estão listadas as tarefas que compõem o projecto e a sua distribuição temporal nas duas fases do estágio em conjunto. Um gráfico com informação mais detalhada do plano para cada uma das fases pode ser consultado no Anexo A.

| Tarefa   | Duração em dias |
|--|-----------------|
| Integração na empresa e estudo de Ruby on Rails                      | 12              |
| Estudo comparativo de <i>frameworks</i>                              | 10              |
| Análise de requisitos e especificação da aplicação                   | 6               |
| Esboço de uma primeira versão da aplicação e migração para Rails 2.0 | 28              |
| Implementação e análise de soluções incluindo testes de usabilidade  | 65              |
| Estudo de mecanismos de <i>caching</i>                               | 15              |
| Escrita de relatório   | 10              |

*Tabela 1 - Distribuição temporal das principais tarefas numa panorâmica geral.*

## 3. Ruby on Rails

---

Ruby on Rails (ou simplesmente RoR) é uma *framework* desenvolvida em Ruby com vista à implementação de aplicações web com recurso a uma base de dados, de modo simples e produtivo.

Dando prioridade ao modo simples de fazer as coisas e tirando partido da linguagem Ruby permite uma alta produtividade no desenvolvimento de aplicações web. Meses após o lançamento oficial esta framework passou de desconhecida a uma das frameworks de escolha para implementação de uma variedade das chamadas aplicações Web 2.0, não sendo apenas uma moda junto dos mais entusiastas mas também adoptada por multinacionais para desenvolvimento de aplicações web robustas [2].

A *framework* assenta em duas filosofias chave:

- **Don't repeat yourself (DRY):** reduzir duplicações, particularmente quando se trata de programação. A informação está localizada num único sítio não ambíguo.
- **Convention over configuration (CoC):** significa que apenas temos de especificar aspectos não convencionais da nossa aplicação.

É implementada um arquitectura MVC que ajuda a reduzir a complexidade no *design* da arquitectura das aplicações e aumenta a flexibilidade e reutilização. Assim, existe bastante modularidade e separação das componentes de uma aplicação.

A *framework* está inteiramente pensada para o desenvolvimento de aplicações web, desde a sua concepção, havendo grande focagem na disponibilização de ferramentas que facilitam este tipo de aplicação, como um sistema de *plugins*, *helpers* e migrações de dados.

Todo o estágio envolveu a utilização desta *framework* e das metodologias e tecnologias a ela adjacentes. Durante a primeira fase foi feito um estudo mais aprofundado acerca desta framework que pode ser consultado no Anexo B.

# 4. Estudo comparativo de *frameworks*

---

## 4.1. Introdução

Existem dezenas de outras *frameworks*, algumas precedentes a Ruby on Rails. Neste capítulo descreve-se um estudo comparativo de algumas *frameworks* com maior destaque na presente data. Este estudo teve como inspiração um estudo semelhante realizado para as mesmas *frameworks* [3] e tem como principal objectivo avaliar cada uma das *frameworks* ao nível da facilidade de utilização e rapidez de desenvolvimento. Também importa haver abstracção de muitos aspectos que tornam a tarefa de fazer uma aplicação web repetitiva, havendo focagem na criatividade e inovação.

Na altura da realização o autor possuía já conhecimentos em algumas das *frameworks* ou linguagens nele envolvidas (existia já uma experiência prévia bastante aprofundada em PHP, Java, Javascript e algum *know-how* na *framework* Ruby on Rails, resultante do próprio estudo inerente ao estágio). Os resultados devem ser interpretados tendo em conta este conhecimento prévio, no entanto houve um período para a familiarização e experimentação de cada *framework*.

A aplicação implementada é um sistema muito simples de gestão de comentários de utilizadores que deve permitir adicionar, remover e editar utilizadores e os seus comentários. Os dados estão guardados numa base de dados MySQL [4] por ser de fácil instalação e muito comum em aplicações web. A especificação desta aplicação pode ser consultada no Anexo D.

Nas secções seguintes apresentam-se as alternativas analisadas, bem como a experiência no desenvolvimento da aplicação em cada uma das *frameworks*, desde a sua instalação. Em cada alternativa tenta-se evidenciar pontos semelhantes ou divergentes quanto ao desenvolvimento em Ruby on Rails. No final apresenta-se um quadro resumo de quantificação do tempo gasto bem como a quantidade de código escrito de configuração e implementação.



## 4.2. Ruby on Rails

Em Ruby on Rails, com ajuda do mecanismo de *scaffolding*, uma aplicação simples como a deste teste faz-se bastante rapidamente. Correndo um comando de *scaffold* por cada modelo são geradas automaticamente as *views*, as migrações de dados, os controladores e os modelos. Ficam por fazer passos como as validações e pequenas alterações nas vistas para cumprir os requisitos da aplicação.

## 4.3. Django

Django [5] é uma *framework* desenvolvida em Python, uma linguagem bastante semelhante a Perl ou Ruby. A linguagem Python é usada de modo transversal, tal como Ruby é usado na *framework* Ruby on Rails. Tal como Ruby on Rails, foca-se na reutilização e ligação entre os diversos componentes bem como no desenvolvimento ágil e o princípio DRY.

Na instalação da versão utilizada (versão 0.96), contrariamente ao que seria esperado, surgiram algumas complicações, de resolução simples mas não evidente. Para o correcto funcionamento teve de se copiar manualmente algumas bibliotecas. O problema foi resolvido com algumas pesquisas na Internet mas o tempo total de instalação foi de 24 minutos. Depois de feitas algumas experiências e estudado o modo de funcionamento desta *framework* a aplicação de testes foi implementada sem problemas de maior. Não existe aqui um mecanismo para arranque rápido de uma aplicação base como o *scaffold*, pelo que todo o código teve de ser escrito de raiz e dada a pouca experiência em Python isso apresentou uma dificuldade acrescida, tendo-se cometido muitos erros, como seria de esperar.

Esta *framework* tem incluído um servidor de testes que permite independência no desenvolvimento da aplicação. Existem também ferramentas de linha de comandos que permitem facilidades de desenvolvimento como a validação de modelos e testes unitários. A criação das tabelas na base de dados é feita de modo automático pela análise da definição dos modelos, pelo que o tempo dispendido na escrita de SQL e interacção directa com a base de dados é nulo. A tarefa de criação dessa base de dados totalizou pouco mais de 5 minutos e a dificuldade foi apenas saber que tipos atribuir a cada campo dos modelos uma vez que o nome não é intuitivo. As

validações dos campos é automática uma vez que em Django é assumido que todos os campos são obrigatórios por omissão.

Django tem a grande vantagem de criar uma interface de administrador, quase automaticamente (implica apenas adicionar mais três linhas de código). Na Figura 3 está uma secção dessa interface, onde se pode ver o aspecto bastante cuidado e o mapeamento automático dos parâmetros de cada modelo.

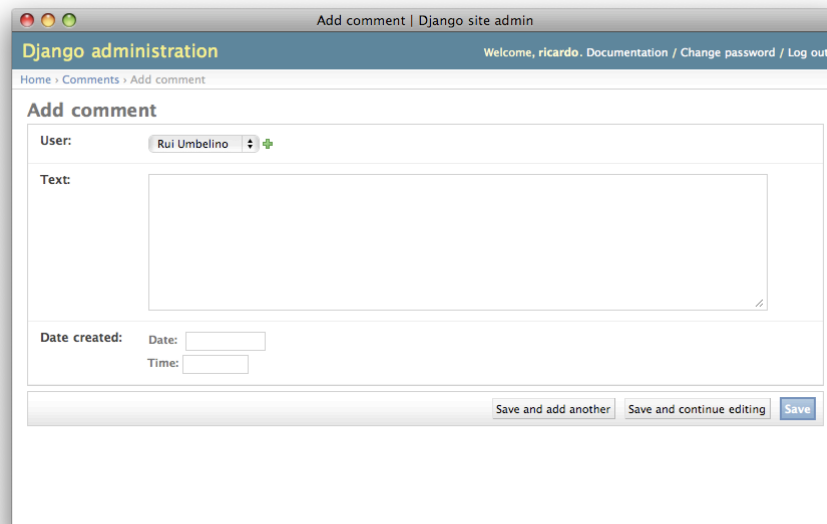


Figura 3 – Página de adição de um comentário no backoffice de administração em Django.

Como se pode ver na figura esta interface é bastante completa e pode ser refinada com algum trabalho adicional. Além disso tem autenticação já incluída, sendo aceitável a utilização deste tipo de mecanismo quando em ambiente de produção.

O desenvolvimento da aplicação fez-se num total de 30 minutos, onde as dificuldades encontradas foram muitas vezes dependentes da pouca experiência com a *framework*. Um aspecto que causou alguma confusão inicialmente foi a organização de um projecto: dentro de um projecto criam-se aplicações (neste caso a aplicação era a gestão dos utilizadores e comentários). Um projecto pode conter várias aplicações e as aplicações podem pertencer a diversos projectos, funcionando como *plugins*. A maior flexibilidade na estruturação de uma aplicação adiciona complexidade à sua implementação.

Django é uma *framework* sólida e de desenvolvimento rápido e apenas ficam a faltar aspectos como uma maior biblioteca de *helpers* para a construção das *views* e um mecanismo de migrações. Esta *framework* poderá no entanto ser ideal em casos onde:

- seja necessária a presença de uma interface de administração completa de modo rápido e fácil;
- existe *know-how* na linguagem Python;
- há preferência por uma filosofia de código explícito e não de convenção sobre configuração e código conciso;

## 4.4. Zope+Plone

Zope (*Z Object Publishing Environment*) [6] é um servidor de aplicações web desenvolvido totalmente em Python e é um poderoso ambiente de programação web, orientado a objectos, que permite desenvolver aplicações remotamente. As aplicações de uso do Zope são as mais diversas, porém tem sido utilizado em larga escala por diversas corporações em aplicações de CMS.

As aplicações criadas no Zope são chamadas de produtos, sendo Plone [7] um dos produtos mais conhecidos em aplicações web públicas. Plone é uma *framework* de gestão de conteúdos e apresentado como um dos melhores CMS *Open Source*, tendo sido em grande parte impulsionado pela sua utilização na NASA.

A instalação destas duas tecnologias foi, de longe, a mais atribulada e foram precisas cerca de três horas até que tudo estivesse a funcionar correctamente. Primeiro teve de se compilar Zope a partir do código fonte e para a versão instalada (versão 2.10.6) há dependência da versão 2.4.5 de Python obrigando a um *downgrade* uma vez que a versão de Python instalada era mais recente. Depois de se ter Zope compilado e instalado procedeu-se à instalação de Plone (versão 3.1.1) que obriga por sua vez à compilação de mais de 4 bibliotecas diferentes para funcionar.

Outro problema encontrado é que em Zope por defeito trabalham-se com dados em ZODB (*Zope Object Database*), uma camada persistente de objectos em Python, não incluindo suporte para MySQL. Assim, de modo a cumprir os requisitos da aplicação, é necessário proceder à instalação de bibliotecas adicionais para gestão de dados em MySQL através de um produto no Zope. A definição das tabelas tem de ser feita interagindo-se directamente com a base de dados obrigando à escrita de todo o SQL em separado.

Finalmente instalada toda a *framework*, pode ter-se acesso a uma interface web de administração quer do Zope quer de todos os produtos instalados. Esta interface (ver

Figura 4) publicitada como sendo amigável e de utilização simples é, na verdade, extremamente complexa, imbricada e pouco intuitiva para quem, como o autor, não está habituado ao desenvolvimento de aplicações usando este tipo de aproximação.

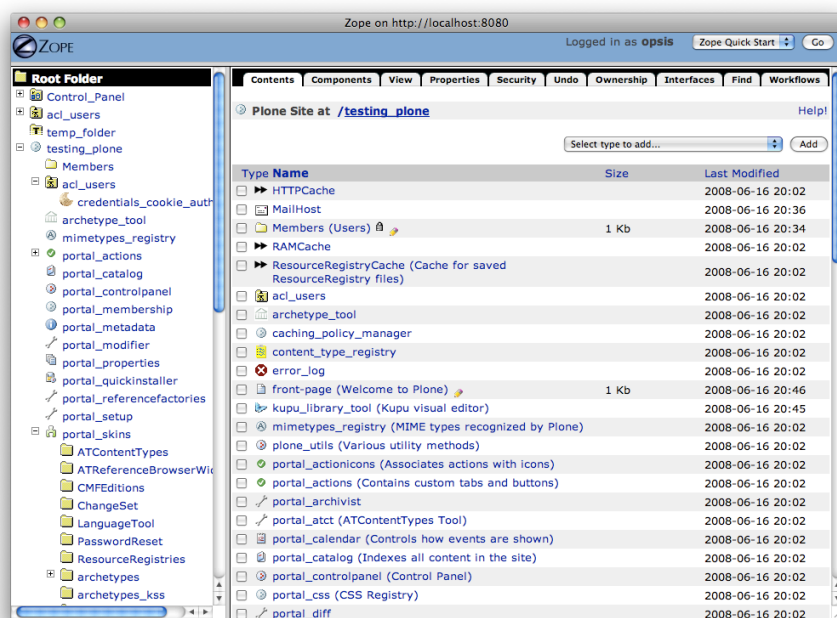


Figura 4 – Aspecto geral da página de administração de um projecto usando Zope+Plone.

Como se pode ver pela figura existe uma extensa listagem de parâmetros a configurar com base numa aplicação pré-instalada e funcional. É verdade que quase todos os aspectos da aplicação podem ser configurados mas houve muita dificuldade na personalização da aplicação, uma vez que os menus de configuração são imensos e requerem muita habituação. A juntar à dificuldade de os menus serem muitos vêm ainda os nomes pouco intuitivos ou esclarecedores de cada componente ou acção, que conduzem a configuração a um ciclo de tentativa/erro.

O desenvolvimento da aplicação não foi agradável e muitos aspectos que se gostaria de ver modificados no resultado final (por não serem necessários) tiveram mesmo de permanecer, uma vez que se perdeu cerca de duas horas no desenvolvimento de uma aplicação tão simples e as tentativas de a personalizar levaram a erros pouco esclarecedores e quebra nas dependências. O desenvolvimento e personalização usando exclusivamente o *browser* não se revelou por isso vantajoso, sentindo-se falta de um editor mais completo e os possíveis benefícios da interacção directa com a aplicação ao longo do desenvolvimento são ultrapassados em larga escala pelas dificuldades de manipulação da interface confusa.

## 4.5. CakePHP

PHP é uma das linguagens de *scripting* mais utilizadas no desenvolvimento web pela sua facilidade de utilização. CakePHP [8] é uma *framework* em PHP que não tenta ser uma cópia de Ruby on Rails mas surgiu para dar resposta ao aparecimento desta última implementando muitos dos seus conceitos.

A instalação da última versão disponível (versão 1.1.19.6305) foi feita sem qualquer acidente uma vez que basta fazer o *download* e copiar para uma directoria onde queremos desenvolver a aplicação. Este passo é bastante mais simples que o experienciado nas restantes *frameworks* e fez-se em cerca de três minutos.

Para a definição da base de dados teve de ser escrito todo o SQL interagindo-se directamente com a base de dados. Nenhuma ferramenta nesta *framework* permite a automatização deste processo. Permite, no entanto editar o ficheiro de configurações para ligação a essa base de dados sem que para isso se tenha de o editar directamente, com a ajuda de uma interface em linha de comandos de pergunta e resposta. Nenhuma outra configuração necessita de ser feita, pelo que este passo é muito rápido (máximo de 2 minutos). Na Figura 5 está apresentado parte do *output* desse processo de configuração.

```
Database Configuration:
-----
Name:
[default] > test
Driver: (db2/firebird/mssql/mysql/mysqli/odbc/oracle/postgres/sqlite/sybase)
[mysql] > mysql
Persistent Connection? (y/n)
[n] > n
Database Host:
[localhost] > localhost
Port?
[n] > n
User:
[root] > ricardo
Password:
> ricardotest
Database Name:
[cake] > cake
Table Prefix?
[n] > █
```

Figura 5 – Sistema de interacção com o utilizador na criação de um ficheiro de configuração para ligação à base de dados.

A estrutura de uma aplicação é em tudo semelhante à de Ruby on Rails, existindo convenções rígidas quanto à localização dos diversos componentes bem como à criação do modelo de dados. Isto poderia ser uma dificuldade na familiarização com

a *framework* mas tal não aconteceu uma vez que existia já um *know-how* prévio em Ruby on Rails.

A ferramenta de linha de comandos mencionada em simplifica ainda a criação de modelos, controladores e vistas, de forma interactiva e com base na configuração de base de dados já realizada. Assim, em cinco minutos, sem se escrever qualquer código, é possível definir uma versão básica de toda a aplicação. No entanto o código em PHP das vistas e controladores não é muito agradável de manipular, tornando os ficheiros confusos e pouco sintéticos. Quando se editou os ficheiros de vistas e controladores para maior personalização (por exemplo para apresentar na página de detalhes de um utilizador os comentários a ele associados) perdeu-se mais tempo que no processo de criar todo o esqueleto da aplicação.

Os dados são passados dos modelos para os controladores e para as *views* em simples listas PHP e muita da configuração também é feita usando a mesma técnica. Essas listas podem ser multidimensionais (por exemplo no caso da aplicação de teste o utilizador é uma lista com os diversos parâmetros e pode conter uma lista para cada comentário) e são tratadas de modo explícito, o que além de não ser muito agradável pode gerar alguma confusão visto que a sintaxe não é limpa

Esta *framework* implica a escrita de muito código para funcionalidades simples que vão além das simples operações CRUD geradas automaticamente, o que pode implicar uma manutenção mais difícil. Contudo poderá ser ideal para quem tenha experiência em PHP.

## 4.6. Java EE

*Java Platform, Enterprise Edition* [9] ou Java EE é uma plataforma em Java com bibliotecas que permitem adicionar funcionalidades para o desenvolvimento de aplicações distribuídas, baseadas em componentes modulares, para produção num servidor de aplicações. É composto por uma série de APIs que permitem integrar uma grande variedade de sistemas. No entanto trabalhar com uma *framework* em Java implica conhecer bem as muitas tecnologias envolvidas (Struts, Hibernate, Spring, Axis, etc.), tornando a escolha das tecnologias adequadas a um projecto uma tarefa difícil. Neste estudo optou-se por utilizar Spring [10] com Hibernate [11] e o *deploy* foi feito usando Apache Tomcat [12], por parecer uma solução relativamente simples e paralela às restantes *frameworks* analisadas. A *framework* Spring inclui um

módulo MVC para construção de aplicações web, permitindo adicionar-se *plugins* a qualquer nível da sua arquitectura para suportar diferentes tecnologias.

Inicialmente houve algum tempo para familiarização dos conceitos novos em Java EE e Spring, bem como as alternativas para a implementação de uma solução MVC rápida. A grande diferença sentida quando se utilizou Java EE é que não há uma única solução para o problema e é difícil encontrar duas respostas iguais. O grande número de bibliotecas disponíveis, além de implicar um estudo mais detalhado do estado da arte, obrigou a uma maior familiarização com o próprio funcionamento interno de Spring para que os seus componentes se integrassem correctamente.

A instalação das tecnologias envolvidas decorreu sem problemas, mas juntou-se ao tempo gasto a instalação e configuração do IDE bem como o estudo de como compilar e correr correctamente uma aplicação com estas características. Estas tarefas em conjunto totalizaram 2 horas. O IDE utilizado foi o Eclipse [13], que está bem adaptado ao desenvolvimento em Java EE e permite testar directamente a aplicação, no entanto a sua utilização requer também um certo tempo de aprendizagem.

Não tendo sido encontrada outra alternativa, a base de dados teve de ser criada interagindo-se directamente e escrevendo SQL. É necessários depois definir classes com a ajuda de Hibernate, que podem ser vistas como o modelo, onde se discriminam todas as funções de *set* e *get* e as classes e campos são mapeados directamente para tabelas e colunas com o mesmo nome.

Com a ajuda de um IDE como o Eclipse a escrita do código e das configurações tira partido de detecção de erros à medida que se escreve e de complementação automática de código, que permitem diminuir o tempo perdido em pesquisas na Internet. Apesar de todas as validações aos modelos serem feitas com ficheiros XML, a edição desses mesmos ficheiros pode ser feita de forma simples com ajuda visual, não havendo a necessidade de escrever todo o código à mão. O processo de desenvolvimento, mesmo com as ajudas mencionadas, foi muito complicado para a dimensão da aplicação em questão, tendo-se totalizado quarenta minutos no desenvolvimento desta aplicação. As dificuldades encontradas foram essencialmente em erros de integração dos diversos componentes ou dificuldades com dependências de bibliotecas, que implicaram pesquisas na Internet, com resposta por vezes pouco satisfatória.

## 4.7. Conclusões

Como se pode ver nas secções anteriores, as ferramentas e filosofias utilizadas nas diferentes *frameworks* têm vários pontos de concordância e outros onde são claramente diferentes.

Em Django o código escrito não é tão conciso como em Ruby on Rails uma vez que não assenta na filosofia de convenção sobre configuração. Assim, alguns aspectos podem envolver muito mais código do que em Rails. Como exemplo apresenta-se na Figura 6 uma comparação na definição das rotas para os controladores onde é bem clara a diferença de filosofia destas duas *frameworks*.



```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    (r'^admin/', include('django.contrib.admin.urls')),
    (r'^comments/$', 'user_comments.views.comment_index'),
    (r'^comments/(?P<comment_id>\d+)/$', 'user_comments.views.comment_show'),
    (r'^comments/edit/(?P<comment_id>\d+)/$', 'user_comments.views.comment_edit'),
    (r'^comments/del/(?P<comment_id>\d+)/$', 'user_comments.views.comment_destroy'),
    (r'^comments/new', 'user_comments.views.comment_new'),

    (r'^users/', 'user_comments.views.user_index'),
    ...

ActionController::Routing::Routes.draw do |map|
  map.resources :comments
  map.resources :users

  # Install the default routes as the lowest priority.
  map.connect ':controller/:action/:id'
  map.connect ':controller/:action/:id.:format'
end
```

Figura 6 – Diferença de sintaxe entre o ficheiro de rotas em Django (à esquerda) e em Ruby on Rails (à direita).

A *framework* Zope+Plone foi muito pouco convincente, ficando a sensação de que não está no mesmo patamar que Django ou Ruby on Rails, não se adaptando bem a um ambiente de desenvolvimento rápido. Não é uma plataforma de uso fácil, mesmo no campo das soluções de CMS, havendo alternativas muito mais simples de instalar e personalizar. Prova disso é que todas as aplicações visitadas que foram desenvolvidas com esta *framework* apresentam sempre aspectos em comum com a aplicação configurada por omissão. É realmente difícil, para alguém sem muita experiência, conseguir alterar coisas tão simples como o *design* das páginas criado inicialmente.

CakePHP surpreendeu pela positiva. A criação de um esqueleto para uma aplicação faz-se muito facilmente e em muito pouco tempo. No entanto a sintaxe de todo o código é confusa e pode implicar maior esforço de manutenção. Na Figura 7 apresenta-se a diferença entre a definição do modelo *User* e suas validações em CakePHP e RoR evidenciando a utilização de listas de modo extensivo na primeira.



```

<?php
:class User extends AppModel {

    var $name = 'User';
    var $validate = array(
        'first_name' => array('minlength', 1),
        'last_name' => array('minlength', 1),
        'phone' => array('minlength', 1)
    );

    var $hasMany = array(
        'Comment' => array('className' => 'Comment',
            'foreignKey' => 'user_id',
            'dependent' => false,
        )
    );
}
?>

```

```

class User < ActiveRecord::Base
  has_many :comments

  validates_presence_of :first_name
  validates_presence_of :last_name
  validates_presence_of :phone
end

```

Figura 7 – Diferenças na definição do mesmo modelo de utilizador em CakePHP (à esquerda) e Ruby on Rails (à direita).

Apesar de Ruby on Rails ser uma *framework* nova e excitante para muitos, o *core* da arquitectura segue os padrões presentes em Java EE. A filosofia de desenvolvimento de aplicações web é o que diferencia estas duas *frameworks*. Rails dá primazia a código explícito ao invés de ficheiros de configuração e a natureza dinâmica do Ruby gera muito código no runtime. Grande parte da *framework* Ruby on Rails foi criada como um projecto único e o desenvolvimento de uma aplicação beneficia de um conjunto de componentes homogêneos. Em contraste, uma *stack* típica de Java EE tende a ser construída de componentes diferentes (um conjunto de bons componentes entre um grande leque) que são geralmente desenvolvidos independentemente e são usados muitos ficheiros XML para os ligar.

O *overhead* inicial de estudo e familiarização com Java EE foi bastante maior do que com as restantes, mesmo depois de já haver um *know-how* significativo em Java ficando sempre a sensação de que poderá existir uma alternativa mais eficaz, dada a quantidade de possibilidades no mundo Java EE. Para se atingir um nível de produtividade semelhante ao de outras *frameworks* exige-se assim muito mais prática e habituação. Como é de domínio muito mais abrangente que as restantes *frameworks* não se encontra tanta ajuda direccionada ao desenvolvimento de aplicações web com recurso a uma base de dados relacional usando as mesmas tecnologias. O maior número de linhas de código nesta *framework* é compensado pela utilização de um IDE muito completo, que permite simplificar o processo de manutenção.

Na Tabela 2 está um resumo da experiência de desenvolvimento da aplicação de teste em termos de número de ficheiros de configuração que é necessário editar,

linhas de código escritas e tempo dispendido na instalação e desenvolvimento da aplicação. É também feita uma apreciação global acerca da quantidade e qualidade de documentação disponível na Internet.

|                           | <b>RoR</b> | <b>Django</b> | <b>CakePHP</b> | <b>Zope Plone</b> | <b>Java EE</b> |
|---------------------------|------------|---------------|----------------|-------------------|----------------|
| Instalação                | 15 min.    | 24 min.       | 3 min.         | 4 horas           | 2 h.           |
| Desenvolvimento           | 5 min.     | 30 min.       | 20 min.        | 2 horas           | 40 min.        |
| Ficheiros de configuração | 1 (+1*)    | 2             | 1 *            | n/a               | 3              |
| Linhas de configuração    | 2          | 14            | 6              | n/a               | 120            |
| Linhas de código          | 12         | 140           | 43             | n/a               | 256            |
| Documentação              | Boa        | Boa           | Razoável       | Má                | Boa            |

*Tabela 2 – Quadro resumo das diferenças na implementação da aplicação de teste.*

*\* - criado de forma automática*

Da experiência adquirida neste estudo pode-se concluir que a escolha acertada para o desenvolvimento rápido de uma aplicação web com as características da maioria das aplicações hoje desenvolvidas está longe de ser Zope+Plone. As restantes podem estar bem adaptadas a diferentes contextos, sendo que RoR e Django se enquadram perfeitamente nos requisitos da empresa. A escolha de migrar os projectos para RoR foi baseada apenas na sua maior rapidez e simplicidade apresentada nos ficheiros de código, que é uma razão válida, visto que não existia também qualquer experiência com Python.

# 5. Roomblick

---

## 5.1. Introdução

*RoomBlick* é um sistema de arrendamento e partilha de quartos desenvolvido no presente estágio, que surge da análise da necessidade de um sistema web de arrendamento de quartos em Portugal, com um conjunto de funcionalidades chave e uma interface agradável, que até aqui não existia. Esta aplicação, além de ter valor para a empresa pela aplicação em si, apresenta um conjunto de requisitos que permite explorar diferentes funcionalidades e ferramentas. Serviu assim como um caso prático de características óptimas ao tipo de estudo pretendido, não só pelos seus requisitos, mas também por permitir bastante flexibilidade, uma vez que não é uma aplicação com prazos rígidos ou clientes definidos. A especificação da aplicação pode ser consultada no Anexo E.

Utilizando-se o *RoomBlick* como caso prático fez-se integração com OpenID para a autenticação no sistema e com *Google Maps* para georreferenciação dos quartos; implementou-se uma pesquisa avançada eficiente tirando partido de georreferenciação; estudou-se qual a melhor alternativa para a internacionalização e localização comparando-se *plugins* em RoR e testou-se a integração com um *gateway* de SMS. No capítulo 6 está descrito o estudo e integração de diferentes tecnologias e metodologias que apresentam inovação na empresa no desenvolvimento de uma aplicação web em RoR atrás enumeradas. Tendo por base esta aplicação foram ainda analisadas no capítulo 7 metodologias de optimização de aplicações em RoR e metodologias de testes bem como questões relativas à usabilidade de uma aplicação web, que é um aspecto bastante importante e com valor para a empresa.

No final do estágio a aplicação encontra-se funcional mas não final. Algumas funcionalidades mais triviais que as analisadas neste documento não foram ainda implementadas e alterações, fruto de testes de usabilidade, estão a ser estudadas. De resto, o *know-how* adquirido no desenvolvimento desta aplicação tem mais valor do que a aplicação propriamente dita.

## 5.2. Riscos Iniciais

Um projecto de desenvolvimento de um produto com as características do *RoomBlick* acarreta um conjunto de riscos, associados à utilização de tecnologias em constante desenvolvimento, bem como à introdução de conceitos que se revelaram inovadores. Para este caso em particular, a utilização de tecnologias e ferramentas para as quais não existia *know-how* interno até então foi considerado um risco de algum peso. A ocorrência no projecto de um evento relacionado com este risco teria efeitos negativos no seu desenvolvimento, com consequências que poderiam envolver o não cumprimento de todos os objectivos propostos. No decorrer da fase de desenvolvimento, as funcionalidades disponibilizadas pelas *frameworks* em uso revelaram, no entanto, potencial na sua aplicabilidade transversal ao projecto.

## 5.3. Outras soluções

De modo a melhor compreender o estado da arte no que diz respeito a aplicações de arrendamento de quartos em Portugal, foi feita uma análise de aplicações com as mesmas características do *RoomBlick*, identificando-se funcionalidades chave, bem como problemas comuns. As aplicações existentes são extremamente difíceis de utilizar e apresentam muitas falhas em diversos níveis. Descuidam-se também aspectos básicos de usabilidade ignorando-se *guidelines* básicos nesta matéria.

Aliada aos muitos problemas na forma de organizar a informação está a falta de funcionalidades como a georreferenciação, pesquisa e filtragem de quartos, cálculo de distância a pontos de interesse e possibilidade de autenticação com OpenID. O desenvolvimento de uma aplicação com as características do *RoomBlick* é por isso uma mais-valia para os utilizadores que pretendem usufruir de um sistema de arrendamento de quartos com uma interface simples e um conjunto de funcionalidades tão úteis como as referidas. O estudo pormenorizado das soluções semelhantes existentes para o mercado português pode ser consultado no Anexo F.

## 5.4. Implementação

A implementação do *RoomBlick* foi feita em duas fases. No fim da primeira fase, correspondente ao primeiro semestre lectivo, obteve-se um esqueleto simples da

aplicação, contendo apenas funcionalidades básicas como autenticação e registo. Isto porque fazia parte dos objectivos nesta fase o estudo e ambientação com RoR. Na segunda fase do trabalho implementaram-se assim as funcionalidades restantes, havendo mesmo reformulação de trabalho já feito, uma vez que já se possuía melhores conhecimentos. Ao longo de todo o projecto houve o cuidado de analisar alternativas para cada opção tomada, indicando as principais vantagens e desvantagens de cada uma.

A validação dos documentos web é um passo importante que ajuda a garantir a sua qualidade, podendo-se ao mesmo tempo poupar recursos. Por esta razão houve também algum cuidado no *output* gerado pela aplicação para que o resultado final fosse válido, segundo os *standards* definidos pelo *World Wide Web consortium* (W3C) [14] através da ferramenta disponibilizada pelo mesmo [15].

#### **5.4.1. Migração para Rails 2.0 e filosofia REST**

A *framework* Ruby on Rails tem evoluído continuamente desde a sua criação, sendo um projecto bastante activo. Por esta razão teve de haver algum trabalho de adaptação do que estava já desenvolvido (na versão 1.2.3) para uma nova versão lançada em Dezembro de 2007 (versão 2.0). Isto implicou a alteração de algum código já desenvolvido, incluindo o *plugin* para autenticação, uma vez que o que estava a ser utilizado tornou-se *deprecated*, reformulação das migrações feitas para um novo formato mais compacto e tirou-se também partido de um melhoramento significativo da interacção com tecnologia REST.

Assim, a maioria dos controladores usam rotas definidas automaticamente, adicionando-se apenas uma linha ao ficheiro de rotas, indicando qual o recurso que se quer fazer um mapeamento REST. Utilizando este tipo de técnica ajuda também na construção de *views* e redireccionamentos, uma vez que, para cada recurso são disponibilizadas, automaticamente, funções de ajuda. Juntamente com esta nova implementação, tirou-se também partido de *namespaces*, introduzidos na versão 2.0 da *framework*, que permitem definir rotas associadas a recursos agrupados em espaços independentes.

# 6. Integração com tecnologias

---

## 6.1. Introdução

Neste capítulo descreve-se a integração de uma aplicação em RoR com um conjunto de tecnologias. As tecnologias aqui analisadas, ainda que possam não representar uma novidade no mundo das aplicações web em geral, não haviam sido ainda exploradas pela empresa em aplicações RoR, pelo que houve o cuidado de analisar quais as soluções possíveis e compará-las. Ao longo deste capítulo é possível ganhar conhecimentos nas soluções existentes para integração de OpenID para autenticação, georreferenciação, internacionalização e localização de uma aplicação em RoR e integração com um *gateway* de SMS. Para complementar a informação podem ser consultados em anexo documentos que servem de guia prático (ou *howto*) de implementação das soluções para cada problema.

## 6.2. OpenID

OpenID é uma tecnologia livre, que simplifica a experiência da navegação *on-line* eliminando a necessidade de registos múltiplos espalhados por aplicações diferentes. Assim, permite que cada indivíduo possua um registo centralizado com controlo da sua identidade digital. Esta centralização permite que o utilizador deixe de se preocupar com a tarefa incómoda de criar e memorizar os dados de autenticação em dezenas, ou mesmo centenas, de sítios diferentes.

O seu funcionamento está descrito na Figura 8. Um utilizador com conta criada num dos vários facilitadores existentes pode utilizar o seu endereço para fazer o *login* na aplicação.

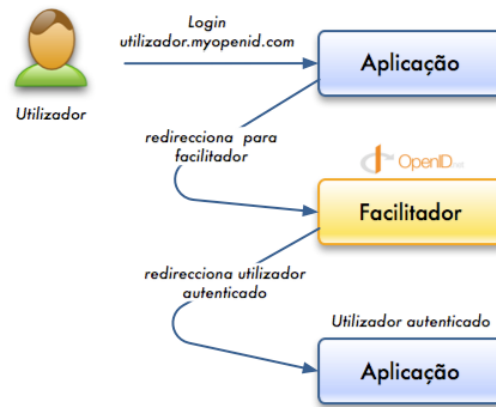


Figura 8 – Funcionamento da autenticação usando OpenID.

Convém referir que quando a aplicação comunica com o facilitador há troca de chaves para fiabilidade do sistema, uma vez que a aplicação só vai depois aceitar um utilizador válido se a chave coincidir com a inicialmente criada.

Esta é uma tecnologia cada vez mais comum, (hoje mais de 10.000 *sites* diferentes permitem a utilização de OpenID para autenticação e estima-se que existam já cerca de 350 milhões de utilizadores que recorrem a este sistema), tendo sido recentemente apoiada por grandes empresas como a Microsoft, o Google e o Yahoo! [16].

### 6.2.1. Implementação

Não havendo na empresa nenhum tipo de experiência com esta tecnologia, e uma vez que, como referido anteriormente, faz todo o sentido preparar as novas aplicações para a sua utilização, estudou-se qual a melhor forma de incorporar um sistema de autenticação com OpenID.

Para fazer o registo de um OpenID pode recorrer-se a diversos serviços já disponíveis [17]. É então disponibilizado um endereço pessoal único que pode ser utilizado para autenticação em aplicações que permitam a sua utilização.

A implementação da autenticação com recurso a um OpenID envolve a criação de uma tabela com associação de utilizadores a OpenIDs, reformulação dos formulários com alternativas ao registo e *login* no nosso sistema e a interacção com o facilitador dos dados OpenID do utilizador.

Para a comunicação com um facilitador OpenID existe já implementada uma biblioteca [18] que evita a perda de tempo com o processo que poderia ser o mais

complexo na integração de OpenID com a nossa aplicação. Para a interação com o utilizador (*views* e *controllers* no Rails) existem duas alternativas em forma de *plugin* e como tal não é necessário desenvolver a funcionalidade de raiz. Esses *plugins* são:

- “*openid\_consumer*” – desenvolvido pela EastMedia para a VeriSign [19];
- “*open\_id\_authentication*” – desenvolvido pela equipa do Ruby on Rails [20].

Depois de alguma experimentação chegou-se à conclusão que a integração do “*open\_id\_authentication*” é um processo muito menos trabalhoso, uma vez que utiliza os mesmos controladores que o *plugin* de autenticação que estava a ser utilizado (também da equipa do Ruby on Rails). Além disso tem na sua documentação diversos exemplos da implementação dos dois *plugins* em paralelo.

O primeiro *plugin*, no entanto, funciona de modo semelhante e poderá ser utilizado caso a autenticação seja feita de outra forma (ou se só é utilizada autenticação por OpenID), uma vez que não está intimamente associado a qualquer outro *plugin*. Este gera apenas o controlador para comunicação com o facilitador de OpenID e um ficheiro de migração para criar a tabela de associação dos OpenID’s com os utilizadores.

No Anexo G podem ser consultados pormenores práticos relativos à implementação que pode ser consultado como um *howto* na integração de um sistema de autenticação com OpenID em Ruby on Rails.

### 6.2.2. Conclusões

A inclusão de um sistema de autenticação com OpenID é, como vimos, facilitada pela *gem* e o *plugin* em conjunto, e pode ser uma mais-valia para qualquer aplicação. No entanto, até ao momento, a prática comum é utilizar este sistema como alternativa e não como único, uma vez que ainda não está suficientemente difundido. Além disso o processo pode levar a alguma habituação por parte dos utilizadores, ou poderá mesmo não ser do agrado de outros, uma vez que adiciona complexidade ao processo de autenticação. Por esta razão, normalmente é adicionado como alternativa e não como sistema único de autenticação. Na Figura 9 apresenta-se a solução encontrada para dar o *login* com OpenID como alternativa.





Figura 9 – Alternativa de login com OpenID na aplicação RoomBlick.

Resta ainda alertar para as possíveis falhas de segurança deste sistema. A versão actual deste tipo de autenticação está bastante sujeito a ataques de *phishing*, uma vez que, como já foi referido, quando o utilizador envia o seu endereço OpenID não estando ainda autenticado no facilitador é-lhe pedida a *password* para autenticação, podendo neste caso ser um facilitador falso.

## 6.3. Georreferenciação

Georreferenciar é tornar conhecidas as coordenadas de algo num dado sistema de referência. Este processo inicia-se com a obtenção das coordenadas (pertencentes ao sistema no qual se pretende georreferenciar) para um ponto de referência que oferece uma feição física perfeitamente identificável, tal como intersecções de estradas e de rios, represas, pistas de aeroportos, edifícios proeminentes, topos de montanha, etc.

A informação espacial pode ser de grande importância em determinadas aplicações, havendo por isso necessidade de apresentá-la e manipulá-la da melhor maneira. No caso específico do *RoomBlick*, a possibilidade de visualizar a localização do quarto num contexto global e calcular distâncias a pontos de interesse seria mesmo um requisito fundamental e que distingue esta aplicação das já existentes.

### 6.3.1. Pontos de interesse

Na aplicação em causa, um dos parâmetros considerados como fundamentais seria a proximidade a pontos de interesse (como escolas, estações de comboio, etc.). Assim, explorou-se qual a melhor forma de integrar tecnologia de georreferenciação para disponibilizar informação mais detalhada da localização dos quartos bem como a inserção dos referidos pontos de interesse. Os pontos de interesse são inseridos por

um administrador, individualmente ou fornecendo ficheiros com listas de pontos de interesse. Existem listagens de pontos de interesse já agregados disponíveis na Web [21] para utilização em sistemas de GPS. O único problema aqui encontrado foi a quantidade enorme de formatos lidos pelos dispositivos GPS. Não havendo um *standard* para esta informação optou-se pela solução mais fácil (ficheiros CSV) que é, de resto, a utilizada por alguns fabricantes. De qualquer modo existem programas [22] que podem converter qualquer formato utilizado para CSV, tornando-se assim possível inserir, de forma fácil, as inúmeras listas de pontos de interesse.

### 6.3.2. Apresentação e cálculo de distâncias

No que diz respeito à apresentação da informação geográfica, existe um leque de diferentes alternativas disponíveis, que permitem integração noutras aplicações. A escolha aqui caiu sobre a alternativa mais rápida, uma vez que todas cumprem os requisitos necessários à apresentação simples de pontos georreferenciados. Assim, para a apresentação da localização geográfica de um quarto utilizou-se o Google Maps [23], serviço pioneiro neste campo. Mais há a acrescentar que, além de ser o serviço mais rápido [24] é o que existe há mais tempo, havendo assim maior familiarização com a interface disponibilizada e também permite pesquisar coordenadas com base num endereço, útil para a inserção de um novo quarto, como está descrito na secção seguinte. Na Figura 10 está um exemplo da apresentação geográfica de um quarto.



Figura 10 – Localização geográfica de um anúncio usando a API do Google Maps.

Para o cálculo de distâncias utilizou-se um *plugin* – GeoKit [25] – que, de entre os existentes é o que tem mais funcionalidades e apresenta maior número de funções úteis e permite recorrer a *Geocoders* múltiplos (Google, Yahoo! etc.). Outra razão que leva a usar este *plugin* é a capacidade de devolver a localização baseada num IP, importante para a questão de internacionalização com definição automática da língua (ver secção 6.4), funcionalidade de grande importância ao nível de usabilidade (este tópico será analisado no capítulo 7 – secção 7.4).

### 6.3.3. Pesquisa por morada

Os quartos são adicionados usando um mecanismo diferente do adoptado relativamente aos pontos de interesse. As coordenadas têm de ser definidas pelos utilizadores e não é razoável obrigar a sua inserção directa quando é adicionado um novo anúncio.

Desta forma, tirando partido da API do Google Maps, é possível permitir que o utilizador forneça a morada do imóvel, obtendo-se as coordenadas do mesmo automaticamente e apresentando essa informação visualmente. Um exemplo de inserção visual das coordenadas de um quarto pode ser visto na Figura 11.



*Figura 11 – Ajuda visual para adição das coordenadas de um quarto. A marca está a ser arrastada pelo utilizador, para refinar a localização.*

Este método permite ainda mover a localização do quarto, bem como aproximar ou afastar a vista do mapa, sendo assim um método transparente e de utilização

amigável para a definição da latitude e longitude do quarto. O único potencial problema aqui é a possibilidade de se fornecer moradas que não conseguem ser encontradas ou mal formatadas. Endereços ambíguos podem levar a problemas de interacção com o utilizador, sendo por isso por vezes dada alguma ajuda quanto ao formato da morada a pesquisar.

#### 6.3.4. Exportação de dados

Dada a riqueza de informação contida na aplicação relativamente a quartos para arrendar e sua localização, é interessante poder partilhar-se essa informação para integração noutras aplicações, não se criando um silo de informação. Neste contexto explorou-se uma aspecto importante da *framework* Ruby on Rails – a facilidade de exportação de dados.

Assim, como exemplo, e usando rotas REST anteriormente definidas, disponibiliza-se em formato válido para importação no Google Earth [26] (ficheiros KML) a informação de um quarto ou da listagem de quartos disponíveis em determinada região. Na Figura 12 apresenta-se o mesmo exemplo da Figura 10 quando importado, para o Google Earth. Esta informação é actualizada automaticamente e sincronizada com as alterações na aplicação web.

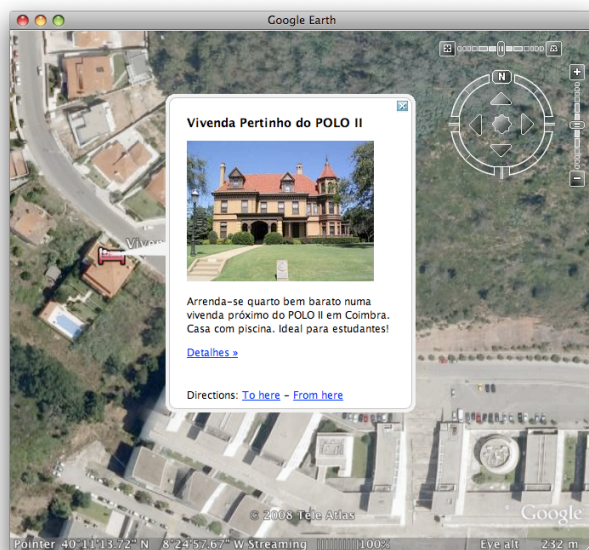


Figura 12 – Informação geográfica de um quarto usando o Google Earth.

Estes dados podem ser, de modo semelhante, exportados para uma variedade de outras aplicações, sendo útil para maior flexibilidade de utilização da informação. Os

utilizadores podem desta forma, por exemplo, calcular o roteiro de suas casas até ao imóvel em questão. Pode fazer parte de trabalho futuro a disponibilização desta informação, servindo a exportação para o Google Earth como exemplo.

### 6.3.5. Conclusão

A disponibilização de informação georreferenciada na Web tem vindo a evoluir bastante nos últimos anos. Encontram-se aplicações com georreferenciação de crimes, fotografias e bombas de gasolina. Quase tudo é objecto de ser localizado num mapa. A evolução dos sistemas de mapeamento, disponibilizando APIs cada vez mais completas, permitiu que o mapeamento de informação fosse uma funcionalidade muito popular na Internet de hoje.

A disponibilização dessa informação faz tanto mais sentido quanto mais o objecto tratado pela aplicação está intimamente relacionado com uma localização geográfica. Disponibilizar essa informação de forma eficiente, bem como permitir manipulá-la da melhor maneira possível é, de resto, uma mais-valia de peso em qualquer aplicação.

A integração com o Google Maps implica um estudo da API disponibilizada pelo Google, que, sendo uma biblioteca bastante complexa de Javascript, obriga a algum tempo de familiarização. Com a utilização do GeoKit algumas tarefas que poderiam ser complexas ficam bastante simplificadas, como o cálculo de distâncias entre pontos georreferenciados. No Anexo H pode ser consultado um guia prático com o modo de como foi implementada a georreferenciação, a exportação dos dados e detecção automática da localização com base no IP para a aplicação *RoomBlick*. Este documento pode ser visto como um *howto* na implementação deste tipo de funcionalidades em RoR.

## 6.4. Internacionalização e localização

Internacionalização e localização são tópicos importantes nas aplicações web modernas, e importa distinguir os dois termos:

- **Internacionalização** é permitir que a aplicação aceite, processe e devolva textos em diferentes línguas.

- **Localização** é disponibilizar essa aplicação personalizada para um determinado local (por exemplo formatos de números e datas).

A localização de uma aplicação web é muito diferente da sua internacionalização, no entanto o segundo é pré-requisito do primeiro. Quando se fala em localização, normalmente significa apresentar ao utilizador uma interface diferente (normalmente apenas textualmente) com base no local escolhido.

#### 6.4.1. Internacionalização e localização em Rails

Um dos pontos negativos apontados à *framework* Ruby on Rails é a falta de suporte para internacionalização e localização. Para colmatar esta falha, têm surgido diversas alternativas, havendo já dezenas de *plugins* para Ruby on Rails que permitem “internacionalizar” uma aplicação [27].

Como recentemente a *framework* sofreu um *upgrade* para a versão 2.0 analisaram-se aqueles actualizados mais recentemente e os que tinham mais funcionalidades, uma vez que estariam mais bem adaptados a qualquer aplicação futura. Assim, a lista de alternativas analisadas foi a seguinte:

1. **Simple Localization** - tenta tornar a tarefa de internacionalização o mais simples possível, utilizando ficheiros YAML, muito usados em Rails. Suporta poucas línguas à partida mas é possível definir novas também por edição de um ficheiro YAML.
2. **Gettext Localize** - assenta no gettext - biblioteca de internacionalização *open source* [28] e por isso depende da *gem* “*ruby-gettext*”. Dispõe-se assim de um conjunto de ferramentas para todas as plataformas bem como documentação detalhada para a sua utilização.
3. **Localization Plugin** - este *plugin* utiliza um método semelhante ao *gettext* para as traduções. As traduções são guardadas num ficheiro Ruby e permite que se defina funções dinâmicas em código Ruby para cada entrada. Assim pode definir-se desde formatos de data até plurais da língua que se quer definir. Por omissão não tem qualquer língua definida. É um *plugin* muito simples e leve.
4. **Globalize** - todas as traduções são gravadas na base de dados. Pode traduzir não só texto inserido nas *views* mas também texto guardado na base de dados.

5. *GlobaLite* - o nome deriva do *plugin* anterior mas foi feito um esforço para manter o processo mais simples e sem recurso à base de dados.

Na Tabela 3 é apresentado um resumo das funcionalidades identificadas como mais importantes para cada um dos *plugins* acima mencionados. As funcionalidades são as seguintes:

1. Extracção automática das *strings* - existência de um método automático de recolha dos excertos que se querem traduzidos;
2. Tradução dos modelos - existência de um método de tradução dos nomes dos modelos e dos seus atributos;
3. Tradução automática de erros e *helpers* - possibilidade de ter as mensagens de erro associadas a uma operação num modelo traduzidas, bem como o formato de diversos *helpers* presentes na *framework* (ex: formato das datas e horas, números, etc.);
4. Suporte para plurais - existência de um método para tradução de plurais (à semelhança do *helper pluralize* existente no Rails);
5. Línguas suportadas por omissão - número de línguas para as quais as funcionalidades dos pontos 2 e 3 já estão implementadas;
6. Suporte nativo para português de Portugal - inclusão ou não de português de Portugal no ponto 5;
7. Guarda mensagens em - formato em que fica guardada a informação para a tradução dos textos;
8. Aplicação fica dependente do *plugin* - a forma de traduzir implica que removido o *plugin* a aplicação deixe de funcionar tendo de ser reformulada;
9. Cache das traduções - ao arrancarmos a aplicação as traduções ficam guardadas em memória, não sendo possível alterar nada em *runtime*.
10. Documentação - quantidade e qualidade de documentação encontrada na Internet, bem como a qualidade dos exemplos e ajudas que são instalados com o *plugin*.

|                                  | Simple Localization | Localization Plugin | Gettext Localize | Globalize | Globalite |
|----------------------------------|---------------------|---------------------|------------------|-----------|-----------|
| Extracção automática das strings | Não                 | Não                 | Sim              | Sim       | Não       |
| Tradução dos modelos             | Não                 | Não                 | Sim              | Sim       | Não       |

|   |                        |                      |                     |               |                        |
|---|------------------------|----------------------|---------------------|---------------|------------------------|
| Tradução automática de erros e helpers    | Sim                    | Não                  | Sim                 | Sim           | Sim                    |
| Suporte para plurais                      | Não                    | Não                  | Sim                 | Sim           | Sim                    |
| Línguas suportadas por omissão            | 8 línguas              | Nenhuma              | 24 línguas          | 7599 línguas  | 11 línguas             |
| Suporte nativo para português de Portugal | Não                    | Não                  | Não                 | Sim           | Sim                    |
| Guarda mensagens em                       | Ficheiros YAML (.yaml) | Ficheiros Ruby (.rb) | Ficheiros (.po/.mo) | Base de dados | Ficheiros YAML (.yaml) |
| Aplicação fica dependente do plugin       | Sim                    | Não                  | Não                 | Sim           | Sim                    |
| Cache das traduções                       | Não                    | Sim                  | Não                 | Sim           | Sim                    |
| Documentação                              | Muito boa              | Má                   | Muito boa           | Boa           | Boa                    |

Tabela 3 – Quadro comparativo de alguns plugins de internacionalização.

Esta tabela surge da experimentação de cada um dos *plugins* numa aplicação muito simples mas onde se pôde analisar todos os aspectos apresentados. Esta aplicação apresenta uma listagem como a da Figura 13, de modelos inseridos numa base de dados comum a todos os *plugins*. Esta inserção permitiu testar a internacionalização de erros de cada um dos *plugins*.

| Listando páginas                      |                      |                                |   |
|---------------------------------------|----------------------|--------------------------------|---|
| Tem 4 páginas                         |                      |                                |   |
| Listagem das páginas na base de dados |                      |                                |   |
| Título                                | Corpo                | Criada em                      |   |
| Página um                             | Página número um     | 30 de Maio de 2008 às 04:10    | <a href="#">Mostrar</a> <a href="#">Editar</a> <a href="#">Apagar</a> |
| Página dois                           | Página número dois   | 19 de Março de 2008 às 18:16   | <a href="#">Mostrar</a> <a href="#">Editar</a> <a href="#">Apagar</a> |
| Página três                           | Página número três   | 10 de Abril de 2008 às 20:40   | <a href="#">Mostrar</a> <a href="#">Editar</a> <a href="#">Apagar</a> |
| Página quatro                         | Página número quatro | 01 de Janeiro de 2008 às 11:00 | <a href="#">Mostrar</a> <a href="#">Editar</a> <a href="#">Apagar</a> |
| <a href="#">Nova página</a>           |                      |                                |   |

Figura 13 – Vista principal da aplicação de teste dos plugins de internacionalização.

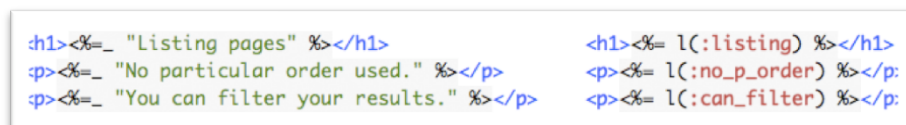
A inclusão por omissão de métodos de tradução automática para os diversos *helpers* presentes em Ruby on Rails é muito importante, uma vez que pode ser um processo complexo e moroso. Assim, o *Localization Plugin*, apesar de muito simples e de fácil utilização, implica bastante trabalho adicional, não tendo sido encontrados exemplos



completos na Internet. No entanto, é o mais flexível uma vez que tudo é definido em Ruby pelo programador.

Outro aspecto importante e que diferencia muito o tempo que uma tradução consome a fazer e manter é a presença de um mecanismo de extracção automática dos textos a traduzir. Os *plugins* que não têm este tipo de mecanismo obrigam que o tradutor tenha de escrever todas as entradas, uma a uma, num ficheiro. Este processo pode ser aceitável para uma aplicação simples (com poucos textos traduzidos) mas com mais de poucas dezenas de entradas este tipo de edição torna-se muito pouco eficaz, principalmente numa fase de desenvolvimento, em que os textos mudam constantemente.

É também importante que, nos templates HTML, seja fácil de perceber qual o conteúdo, o que só acontece nos *plugins Globalize* e *Gettext Localize*. Na Figura 14 apresenta-se um excerto de uma vista evidenciando a diferença entre a utilização de um dos referidos *plugins* e uma das variantes (*Simple Localization*) onde as vistas têm difícil leitura, não se percebendo de imediato o que deve figurar em cada tradução.



```
<h1><%= _ "Listing pages" %></h1>           <h1><%= l(:listing) %></h1>
<p><%= _ "No particular order used." %></p>    <p><%= l(:no_p_order) %></p>
<p><%= _ "You can filter your results." %></p> <p><%= l(:can_filter) %></p>
```

Figura 14 – Comparativo de sintaxe nas traduções de *Gettext Localize* (à esquerda) e *Simple Localization* (à direita).

Claro que se poderia usar símbolos com mais caracteres nos *plugins* que não usam os textos directamente nas vistas, mas isso implicava dificultar ainda mais a leitura dos ficheiros de texto onde estão as traduções.

A inclusão de um grande número de línguas e, particularmente do português de Portugal é uma mais-valia, mas todos os *plugins* permitem, de resto, definir novas línguas de modo simples. No entanto o que melhor cumpre esta tarefa é o *Gettext Localization*, uma vez que, assentando no *gettext* permite a utilização de ferramentas visuais próprias e multiplataforma. Na Figura 15 apresenta-se um exemplo de uma dessas ferramentas – Poedit [29], que permite adicionar comentários bem como identificar possíveis más traduções, sendo ideal para um ambiente de edição colaborativo.

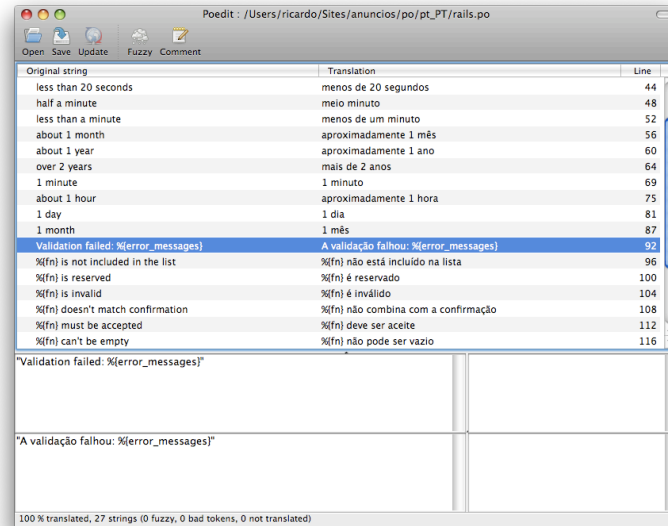


Figura 15 – Janela de edição da tradução de uma aplicação no Poedit.

*Gettext Localization*, é sem dúvida o *plugin* mais completo, não só por ser dos que têm mais funcionalidades, mas porque o modo como são editadas as entradas a traduzir é o mais amigável.

*Globalize* pode ser uma solução fácil para aplicações cujo conteúdo dinâmico (por exemplo notícias inseridas por um administrador) necessitem de tradução para diversas línguas, visto que é o único que permite tradução do conteúdo guardado na base de dados directamente. Claro que, com algum trabalho adicional consegue-se semelhante funcionalidade nos restantes *plugins*.

Utilizando-se a ferramenta de *benchmarking* incorporada no Rails para apurar, na aplicação de teste já mencionada, qual dos *plugins* seria mais rápido e quais as diferenças entre eles. Para isso basta colocar o código da vista dentro de um bloco de *benchmark*, que devolve depois automaticamente qual o tempo gasto para construir esse bloco. Fizem-se 100 *requests* à página de listagem (ver Figura 13) fazendo-se depois a média de tempo de *render* dessa mesma vista. Assim, podemos ver na Figura 16 que o *Gettext Localization* é mais lento que os restantes, por ter o *overhead* da utilização de ferramentas exteriores ao ambiente Ruby on Rails por omissão.

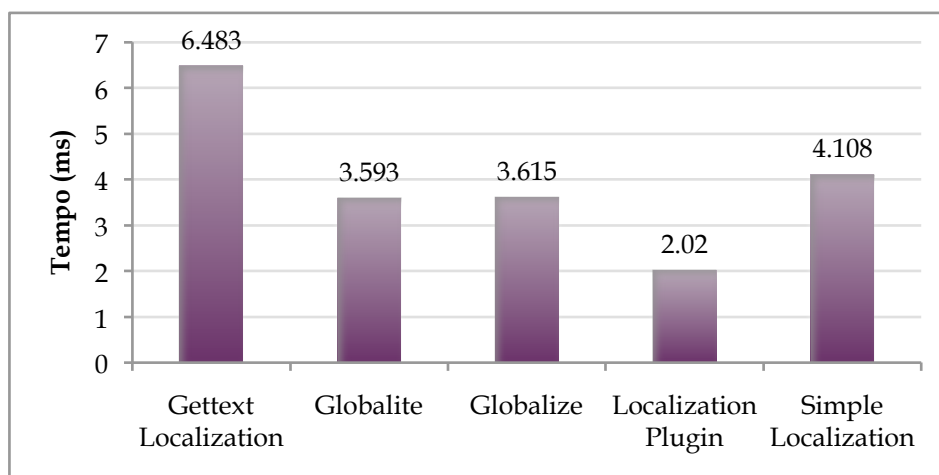


Figura 16 – Tempos médios em milissegundos para o render para diferentes plugins de internacionalização em Ruby on Rails.

O mais eficiente é aquele que utiliza ficheiros em Ruby, carregados no arranque da aplicação, seguindo-se os que carregam as traduções em *runtime* – primeiro os que usam ficheiros de texto, depois o *Globalize* que usa a base de dados. A performance deste último seria afectada, não fosse a utilização de um sistema de *caching* para que não tenha de fazer sempre pedidos à base de dados.

#### 6.4.2. Conclusão

Disponibilizar uma aplicação ao público sem suporte para múltiplas línguas é vedar a sua utilização a milhares de utilizadores. Dependendo do tipo de aplicação isto pode ser, ou não, um problema. No caso concreto da aplicação em estudo, por ser direccionada a público alvo de um único país, não é um problema grave. No entanto, nada impede que existam utilizadores sem conhecimento de português que queiram disponibilizar ou pesquisar quartos em Portugal.

A inclusão de um sistema de internacionalização e localização tem, como foi apresentado neste capítulo, algum trabalho de implementação adicional em Ruby on Rails. Esse trabalho pode mesmo implicar elevados custos de tempo, dependendo da complexidade da aplicação em causa e do número de línguas que se quer cobrir. O problema da tarefa simples de substituição de pequenos blocos de texto é que, de cada vez que se altera um aspecto visual da aplicação implicando alguma mudança de textos é necessário actualizar todas as traduções existentes. Pode chegar ao ponto em que se justifica ter uma equipa de tradução dos conteúdos.

Em Ruby on Rails, a escolha do *plugin* ideal para uma aplicação deve entrar assim em linha de conta com diversos factores, como a importância de uma interface de edição fácil e rápida, a necessidade ou não da mudança dinâmica de traduções e de mecanismos de tradução automática de erros, a necessidade de localização da aplicação e a complexidade ou volume de conteúdos a traduzir.

No caso concreto do *RoomBlick* começou por se utilizar o *Globalize*, por parecer ser dos mais completos, respondendo a necessidades como a tradução de erros e de atributos de um modelo, tendo ao mesmo tempo uma instalação mais facilitada. No entanto este *plugin* tem a desvantagem de, nas *views* não guardar qualquer informação acerca dos textos originais, sendo difícil de se migrar para outro tipo de mecanismo se necessário ou de reconhecer imediatamente os conteúdos de determinada secção. Além disso a tradução é feita interagindo-se directamente com a base de dados. Depois de se experimentar *Gettext Localization* por algum tempo, a clareza das vistas e o controlo de traduções com uma ferramenta visual amigável são pontos suficientemente fortes para contrariar a menor eficiência deste *plugin*.

Importa por último referir que, o *plugin* escolhido também pode estar dependente de quem vai ser responsável pelas traduções. Se a aplicação necessitar de um sistema de *backoffice* na aplicação através do qual se modifica os textos dinamicamente, então não podemos considerar *Localization Plugin* nem *Gettext Localization* uma vez que implicam a reiniciação da aplicação.

O processo de tradução para Português usando *Gettext Localization* pode ser consultado no Anexo I. Este documento pode ser visto como um *howto* de internacionalização e localização em RoR com este *plugin* uma vez que aí está descrito todo o processo prático.

## 6.5. Pesquisa

Um aspecto de extrema importância numa aplicação como o *Roomblick* é a pesquisa de quartos. Este é um dos pontos fracos das aplicações semelhantes analisadas, sendo difícil encontrar um quarto com determinadas características.

Para melhor compreender quais são as prioridades de quem quer arrendar um quarto, fez-se um pequeno questionário, disponibilizado na Internet. O objectivo era apenas ganhar-se sensibilidade para as necessidades dos utilizadores, não se fazendo

assunções de cariz pessoal quanto aos parâmetros mais importantes. Este questionário está disponível no Anexo J.

Como o preço de arrendamento é um parâmetro de grande importância, foi feito um estudo acerca de como organizar uma aplicação que disponibiliza produtos com preços, como agências de viagens e aplicações de *e-commerce* em geral (ver capítulo 7 – secção 7.4). O preço não foi, contudo, considerado como principal parâmetro para a filtragem da pesquisa de quartos, uma vez que, apesar de ser um parâmetro que tem de ter um realce especial por ser importante para quem escolhe um quarto, não serve como um filtro eficiente. Quando uma pessoa tenciona arrendar um quarto, sabe, à partida, qual a zona do país onde quer o mesmo quarto, não fazendo sentido a procura de um quarto em todo o país.

Nesta perspectiva, o segundo parâmetro – localização – é muito mais eficiente numa filtragem. Por esta razão também as aplicações já existentes fazem a filtragem dos anúncios pela localização (ver Anexo F), não sendo possível listar quartos de diferentes regiões em simultâneo – estratégia adoptada no *RoomBlick*.

Associado a essa mesma localização (e porque foi um dos parâmetros que apareceram à cabeça) vem a proximidade a pontos de interesse, e depois ordenaram-se então os restantes parâmetros por ordem de importância, podendo servir de filtros adicionais. Como não são tão importantes são omitidos à partida, de forma a clarificar a interface. A ordem porque aparecem vai ao encontro das necessidades apuradas no inquérito já mencionado. Na Figura 17 pode ver-se um protótipo para esta funcionalidade com os parâmetros de procura à esquerda.

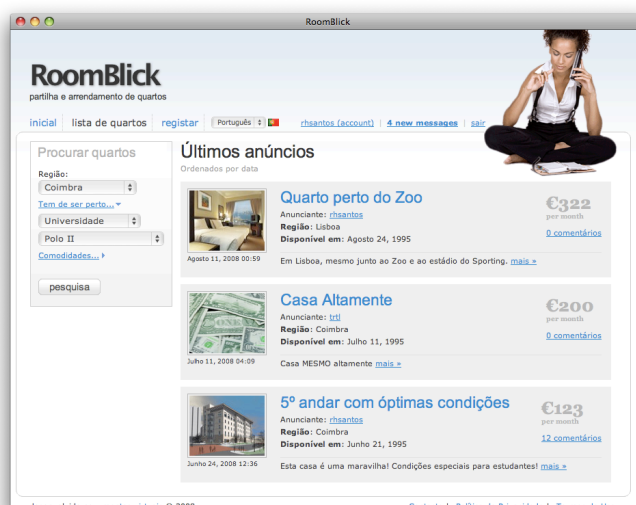


Figura 17 – Página de pesquisa do RoomBlick

Para a pesquisa de quartos com proximidade a pontos de interesse contou-se com o trabalho já realizado no Capítulo 6.3, uma vez que foi utilizado o *plugin* GeoKit para cálculo das distâncias. Esta pesquisa funciona com base na capital de distrito de cada região disponibilizada. Partiu-se do princípio que a inclusão de quartos não pertencentes ao distrito de Coimbra numa pesquisa por quartos perto da Universidade de Coimbra por exemplo, não será um problema, uma vez que os resultados são ordenados por distância ao ponto de interesse.

Uma pesquisa avançada com muitos parâmetros, como a aqui descrita, implica um maior esforço para uma implementação eficiente e de manutenção fácil. A descrição mais detalhada no desenvolvimento desta funcionalidade pode ser consultada no Anexo J.

## 6.6. Integração com *gateway* SMS

Na empresa, está agora em fase de desenvolvimento uma plataforma de serviço de valor acrescentado sobre SMS – PS2 (Plataforma de Serviços SMS) que, usando um *gateway* SMS da PT Inovação, recebe diversos tipos de SMS invocando diferentes serviços.

Um desses serviços, denominado *SMS Ticket* pode enviar por POST os dados recebidos depois de enviada a SMS para o *gateway*. As mensagens são guardadas numa *Message Store* para melhor fiabilidade do serviço, no caso da plataforma PS2 falhar. Na Figura 18 está representado o funcionamento deste serviço e a sua interacção com o *RoomBlick*.

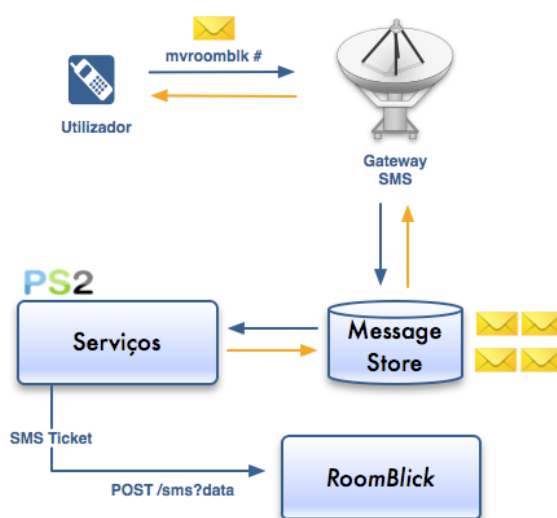


Figura 18 – Funcionamento da plataforma PS2 e integração com a aplicação RoomBlick.

As setas cor-de-laranja indicam a possível resposta por parte do PS2.

A interação com esta plataforma serve de experiência na integração com uma aplicação web em Rails, podendo ser detectados assim possíveis problemas de integração. De resto, a utilização deste tipo de serviço poderá não estar presente na fase final da aplicação, não sendo ainda certa a sua viabilidade. A ser implementado permite assim cobrar um valor a definir por cada anúncio activado, sendo a sua activação bastante simplificada.

Quando um utilizador adiciona um anúncio, este vai estar invalidado, por omissão, não sendo visível para os restantes utilizadores. O ciclo de validação de um anúncio está esquematizado na Figura 19. O anúncio só será realmente apagado depois de um certo período de tempo limite em estado inactivo.

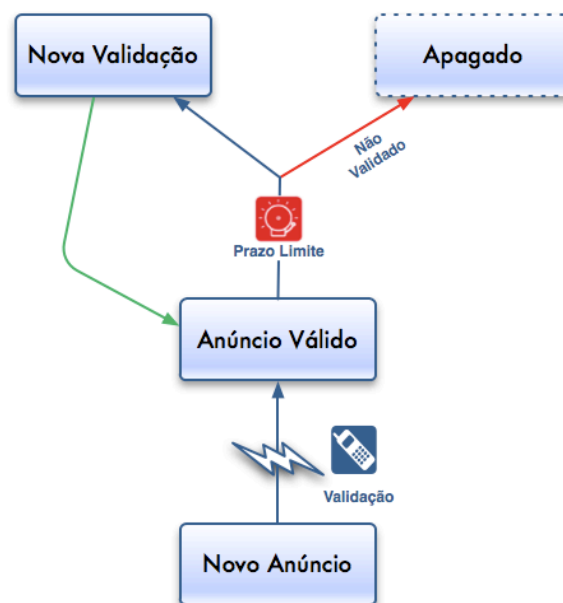


Figura 19 – Ciclo de validação de um anúncio.

Além disso todos os anúncios poderão ter uma data limite, a partir da qual terão de ser activados novamente e deixam de estar visíveis.

### 6.6.1. Troca de mensagens

A mensagem enviada para activação, segundo o serviço PS2, tem de ter uma *keyword*, seguida da mensagem propriamente dita. O serviço sabe automaticamente qual o número remetente e por isso inicialmente pensou-se em tratar o número de

telemóvel como parâmetro obrigatório, podendo assim associar-se um número a um utilizador do *RoomBlick*. Na mensagem só teria de seguir qual o anúncio a activar. No entanto isto traz alguns problemas para o utilizador, que teria de enviar sempre mensagem do mesmo número, tendo de fornecer um número de telemóvel verdadeiro. Isto não agrada os utilizadores (como se pode ver no Anexo M), pelo que houve uma alteração do funcionamento desta activação, permitindo o envio de qualquer telemóvel e não obrigando à inserção de um número verdadeiro.

O serviço PS2 necessita de receber uma *keyword* no início da mensagem para saber qual o serviço para onde deve enviar o pedido. Assim, `mvrroomblk` foi a *keyword* escolhida, seguindo-se do ID do anúncio. Assim, todas as mensagens de activação foram formatadas da seguinte forma:

```
mvrb 2634
```

O URL para onde segue o POST, aquando da recepção da mensagem no PS2, para esta mensagem de exemplo e supondo que vem de um utilizador com o número 916096341, tem como parâmetros:

| Parâmetro | Significado                      | Valor        |
|-----------|----------------------------------|--------------|
| msg       | conteúdo da mensagem             | mvrb 2634    |
| oa        | número de telemóvel do remetente | 351916096341 |
| da        | número do gateway                | 49977        |

*Tabela 4 – Descrição dos parâmetros enviados por POST pela plataforma PS2.*

## 6.6.2. Segurança

Nos dias que correm, as aplicações web correm no ambiente extremamente adverso que é a Internet, e os ataques acontecem, não se podendo confiar na segurança pelo desconhecimento das falhas por parte dos utilizadores.

Enviar um pedido por POST para a aplicação, com o remetente e a mensagem enviada como parâmetros identificando o anúncio do lado da aplicação e activando-o seria um processo bastante inseguro. Se assim fosse qualquer utilizador podia aceder ao URL de activação enviando um pedido por POST activando o anúncio sem qualquer custo. Infelizmente a plataforma PS2 (e principalmente a funcionalidade



em questão) ainda está a ser desenvolvida, não sendo possível garantir que este pedido seja suficientemente seguro.

Desta forma, estudou-se o modo de proteger esta falha, não confiando que os utilizadores não venham a descobrir qual o URL para activação de um anúncio. A solução utilizada temporariamente, que permite proteger o acesso directo de qualquer utilizador ao URL por qualquer pessoa é verificar qual o IP de onde vem o pedido, permitindo apenas o acesso ao IP da aplicação PS2. Esta filtragem é no entanto pouco segura uma vez que qualquer utilizador experiente consegue forjar o IP. No futuro será analisada a questão da segurança em maior pormenor, uma vez que está dependente da aplicação PS2. O ideal, numa versão final seria a utilização de pedidos usando HTTPS e certificados.

# 7. Optimização, testes e usabilidade

---

## 7.1. Introdução

Neste capítulo descreve-se um conjunto de técnicas associadas ao desenvolvimento de uma aplicação em RoR que são fundamentais para se alcançar um resultado final fiável. Assim, começa-se por apresentar os mecanismos de *caching* existentes para a optimização de uma aplicação e as metodologias de testes, essenciais no desenvolvimento usando esta *framework*. A última secção foca-se em aspectos de usabilidade, muito importantes quando se faz desenvolvimento para a Web.

Os dois primeiros capítulos são complementados com anexos onde se descreve a utilização prática destes dois itens servindo de *howto* (ver Anexo K e Anexo L). Para exemplificação prática dos aspectos de usabilidade foram feitos testes onde se analisou os resultados e benefícios em realizar testes desta natureza.

## 7.2. Optimização e caching

Enquanto a construção de uma aplicação web pode não ser um processo muito complexo, construir essas aplicações de modo a serem escaláveis pode ser uma tarefa bastante difícil. As técnicas e tecnologias que funcionam em pequena escala começam a falhar à medida que a aplicação cresce. De modo a evitar-se uma grande perda de tempo e esforço, pensar à partida em problemas de escalabilidade futuros ajuda na construção de aplicações que funcionam bem em pequena escala e que podem crescer até lidar com grandes volumes de tráfego sem necessitarem de reformulações da arquitectura. Numa aplicação web o maior *bottleneck* vai ser geralmente encontrado nos acessos à base de dados [30], usualmente causado pelo I/O de disco. Mas, embora sendo, normalmente, a maior causa de *bottleneck* na aplicação e a primeira causa a receber atenção, é aconselhável identificar outros possíveis problemas.

Uma vez identificado o problema como sendo no acesso à base de dados convém fazer um *query profiling* para identificar que *queries* estão a levar mais tempo e quais são chamadas mais vezes. Aqui, a *framework* pode ajudar, uma vez que permite visualizar essa informação nos *logs* ou em tempo real, quando em ambiente de desenvolvimento. Assim, consegue-se identificar que *queries* estão a atrasar a aplicação chegando-se a soluções para resolver esse problema.

Dada a facilidade de desenvolvimento que RoR proporciona, é comum que se descuidem pormenores relativos à sua optimização. A má performance de uma aplicação web pode ser de tal forma grave que poderá tornar uma aplicação perfeitamente funcional, quando em ambiente de desenvolvimento, completamente impraticável em condições reais [30].

Tendo como base uma versão funcional do *RoomBlick*, analisaram-se assim um conjunto de práticas de optimização. Para ganhar maior sensibilidade quanto à possível optimização de cada uma técnicas usou-se um exemplo da aplicação para comparação em termos concretos. Esse exemplo é uma página de informação sobre um utilizador, que tem também uma listagem de anúncios para esse utilizador, semelhante à da Figura 20.

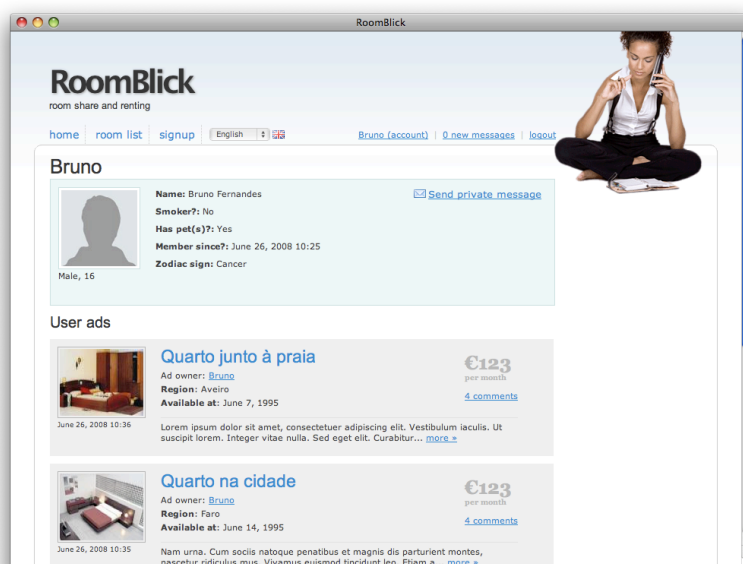


Figura 20 – Vista de teste da aplicação RoomBlick para estudo de opções de optimização.

Este exemplo de teste consiste no menu de navegação, um quadro de informação do utilizador e uma listagem dos seus anúncios, todos de regiões diferentes e cada um contendo 4 comentários. A aplicação corria em ambiente de produção num servidor

*Apache* com *Phusion Passenger* [31] instalado. Todas as medições foram feitas usando a ferramenta *ApacheBench* que vem incorporada com o servidor web *Apache* [32]. Foram feitos a cada configuração 10000 pedidos com 20 pedidos concorrentes:

```
ab -n 10000 -c 20 http://exemplo.com/
```

Para melhor familiarização dos conceitos abordados neste capítulo, é aconselhável a leitura do Anexo B (particularmente o capítulo 10).

No Anexo K é feita uma abordagem prática a todos os mecanismos de *caching* aqui analisados bem como a implementação de *sweepers*, que pode ser consultado para se adquirir *know-how* na optimização de aplicações desenvolvidas em RoR.

### 7.2.1. Counter cache

A primeira linha de optimização de performance da base de dados é a desnormalização. A ideia por trás da desnormalização é muito simples. Certos *queries* consomem bastante tempo, podendo-se poupar esse tempo guardando os dados numa forma não normal, com alguma redundância. Os próprios índices podem ser vistos como uma forma de desnormalização: os dados são guardados num formato alternativo ao longo de uma única linha de dados, permitindo fazer determinadas operações mais rapidamente. A base de dados encarrega-se de manter estes índices sincronizados de modo transparente.

Em Ruby on Rails é muito fácil criar relações do tipo `has_many` definindo um modelo como uma colecção de outro. Por esta razão é comum, e directo perguntar qual o tamanho dessa colecção, por exemplo “*quantos comentários tem este anúncio?*” com chamadas do tipo `anuncio.comentarios.count`. Contudo, esta comodidade pode ser prejudicial, uma vez que, de modo transparente, o que vai ser feito é um `count(*)` na tabela dos comentários, contando o número de linhas referentes ao anúncio em questão.

Apesar de funcionar bem e cumprir o objectivo pretendido, se a chamada for frequente, gera-se bastante SQL extra que seria de evitar. O *Active Record* ajuda a evitar este *overhead* com uma técnica chamada *counter caching*. Na Figura 21 é apresentada uma listagem de anúncios com o número de comentários para cada um



Figura 21 – Caso típico onde faz sentido aplicar counter cache.

Counter caching não é mais que a desnormalização de uma tabela para otimização da performance quando são chamadas funções de contagem. Este processo diminui consideravelmente o número de pedidos à base de dados.

Como se pode ver pelo gráfico da Figura 22, aplicando-se counter caching à página em questão obteve-se uma melhoria de performance em cerca de 10%. Este método não foi aplicado à contagem de mensagens novas que figura no menu de utilizador uma vez que os pedidos são feitos sem a autenticação feita.

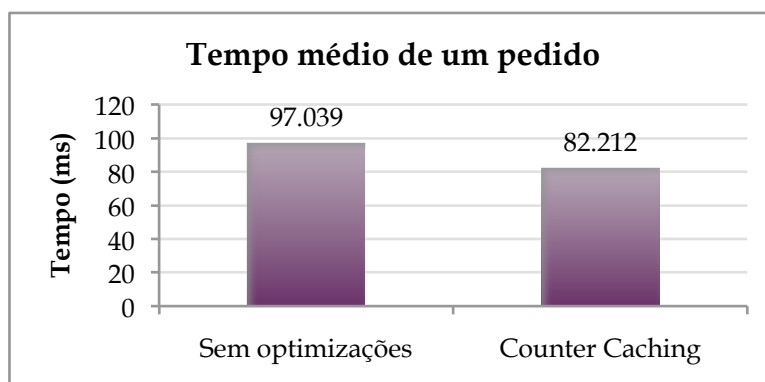


Figura 22 – Tempo médio para pedidos da página de utilizador sem counter caching e com counter caching.

Analisando-se o output do servidor também é possível verificar que há uma diminuição significativa do número de queries à base de dados (potencial ponto de bottleneck). Como se pode ver pelo gráfico da Figura 23 esse número decresce de 32 para 25 pedidos. A diferença será tanto maior quanto mais operações de contagem estiverem a ser feitas.

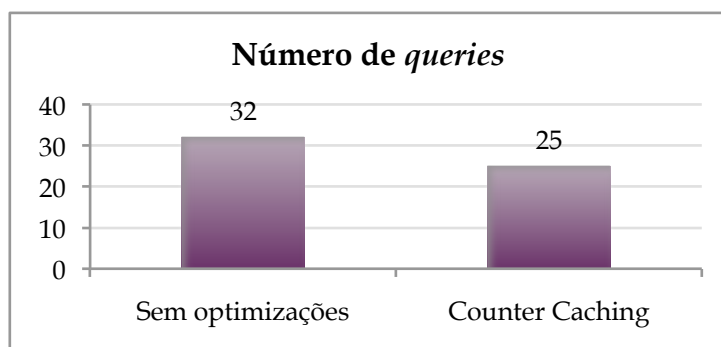


Figura 23 – Número total de queries para apresentar a página de utilizador sem counter caching e com counter caching.

O problema com a desnormalização é que uma base de dados é desenhada para manter os dados sincronizados e consistentes. Se, ao criar-se uma nova coluna `comments_count` o valor correcto não for escrito, então a nossa base de dados fica inconsistente. A *framework* resolve esse problema, usando transacções, para que os dados sejam alterados em sincronia.

### 7.2.2. Action Cache e Page Cache

*Page Caching* e *Action Caching*, na prática, não se enquadram bem com a aplicação em causa, uma vez que em nenhuma altura o conteúdo pode ser guardado na íntegra sem que a cópia guardada fique desactualizada rapidamente. Este mecanismo funciona com base no URL do pedido, guardando uma cópia em HTML desse pedido. Como todas as páginas têm um menu superior dependente de cada utilizador, estes mecanismos não iam justificar-se, havendo maior esforço a guardar cópias e expirar *cache* do que seria desejado.

No entanto, poderiam ser utilizados com alguma mudança no *design*, que pode ou não justificar-se, consoante a possível carga da aplicação. De modo a compreender quais os benefícios destes dois tipos de *caching* efectuou-se o mesmo teste que na secção anterior. Como se pode ver pela Figura 24, através de *Page Caching* é possível obter um aumento significativo na *performance*, uma vez que a página em *cache* é servida automaticamente pelo servidor web, não havendo qualquer interacção com a aplicação em Ruby.

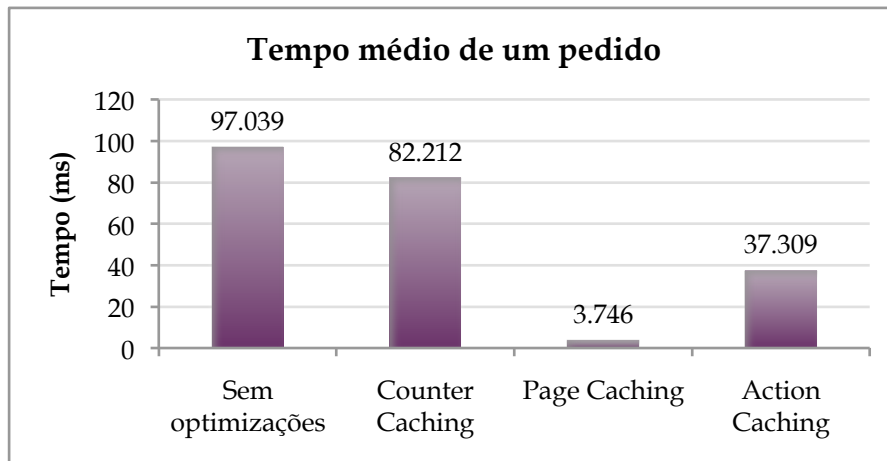


Figura 24 - Tempo médio para pedidos sem otimizações, com counter caching, com page caching e com action caching.

Só o facto de ter de aceder ao código da aplicação para correr possíveis filtros quando se usa *Action Caching* faz com que o tempo médio para servir um pedido aumente substancialmente. Em ambos os métodos não existe qualquer interacção com a base de dados.

### 7.2.3. Fragment cache

Ruby on Rails permite ainda implementar outro tipo de caching, não tão eficiente como os anteriores mas muito útil em qualquer aplicação dinâmica. Com este método consegue ultrapassar-se a dificuldade da utilização de *Page Caching* ou *Action Caching*, uma vez que se guardam apenas fragmentos da página, independentemente do URL do pedido.

No caso particular da página com detalhes de utilizador da aplicação *RoomBlick*, poderia fazer sentido ter em *cache* os detalhes do utilizador e a lista dos seus anúncios ou cada anúncio estar guardado independentemente. Tudo dependia da necessidade real de otimizar estes pedidos e qual a maneira que melhor se adequava. Para efeito de estudo deste mecanismo e sua vantagem dividiu-se a página em 2 fragmentos distintos, como indicado na Figura 25.



Figura 25 – Página de detalhes de utilizador dividida em dois blocos distintos.

Assim, a única componente que permanece sem cópia em *cache* foi o menu de navegação. Os resultados são os apresentados nos gráficos da Figura 26 e da Figura 27.

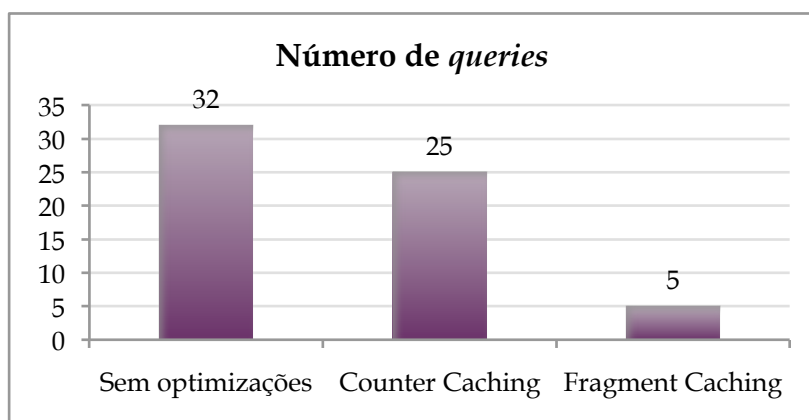


Figura 26 – Número total de queries para apresentar a página de utilizador sem qualquer otimização, com counter caching e com fragment caching.

Como se pode verificar, o número de *queries* de base de dados decresce muito quando se utiliza *fragment cache*, uma vez que maior parte do conteúdo da página vai ser guardado em HTML no disco. Este número pode ser tão baixo quanto se desejar.



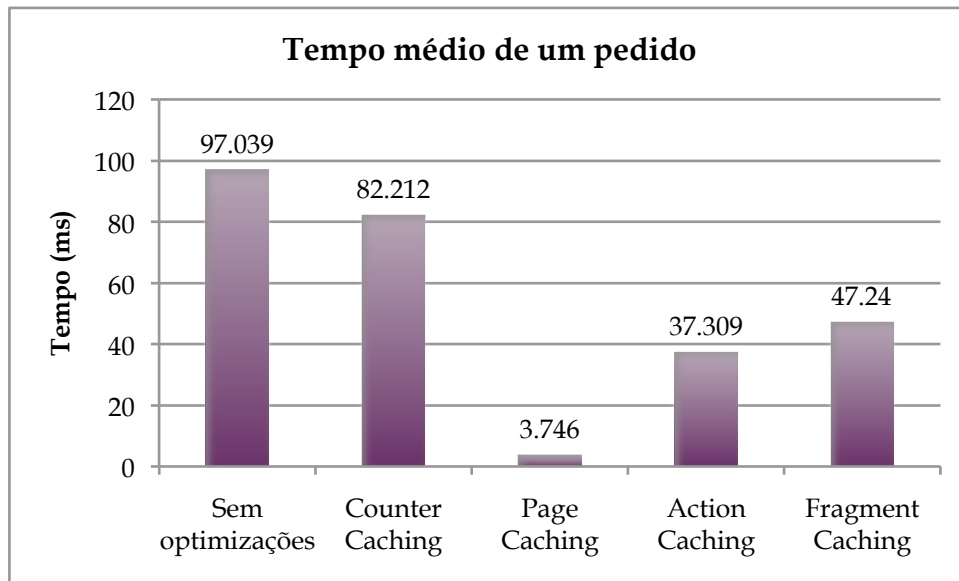


Figura 27 - Tempo médio para pedidos sem qualquer otimização e com cada um dos métodos de caching em Rails.

Quanto ao tempo médio dos pedidos, este método aproxima-se de *Action Caching*, uma vez que este último pode ser visto como um *Fragment Caching* onde o fragmento é a *view* completa. No entanto, *Fragment Caching* é muito mais flexível e útil em aplicações com muitos conteúdos dinâmicos e imbricados.

A única desvantagem deste método é que, como não permite a utilização de *sweepers*, todas as expirações têm de ser explícitas.

#### 7.2.4. Cache em Rails 2.1

Com a versão 2.1 da *framework* (última versão, lançada em Junho de 2008) é possível fazer *caching* de um modo mais generalizado, permitindo guardar em cache qualquer estrutura de dados a partir da nossa aplicação em Rails. Assim, além das *views* passa a ser possível guardar resultados de *queries* SQL ou qualquer código HTML, permitindo um método de *caching* muito mais flexível que os analisados anteriormente.

Guardar os resultados das *queries* SQL, lendo directamente da *cache* em vez de fazer novos pedidos à base de dados, pode ter um grande impacto na *performance* de uma aplicação com muitas leituras da base de dados, e pode ser utilizado em algumas aplicações onde se justifica, mas tem algumas desvantagens. Em primeiro lugar, estamos apenas a guardar em *cache* as leituras, enquanto todas as escritas precisam

de ser feitas directamente para a base de dados e, para manter a sincronia dos dados, os valores em *cache* têm de ser apagados ou actualizados. Isto faz com que o processo de sincronização dos dados esteja do lado da aplicação, adicionando complexidade e possíveis falhas.

Uma alternativa seria adicionar uma camada intermédia entre a aplicação e a base de dados, por onde teriam de passar todas as leituras e escritas, sem qualquer lógica de aplicação. Desta maneira, poderia servir como uma camada genérica e seria mais fácil assegurar que é lida uma cópia válida de dados. A *cache* tem conhecimento de todas as operações de escrita e a base de dados só pode ser alterada com permissão da *cache*. A desvantagem desta solução é a complexidade ainda maior do que a anterior.

Com o novo sistema de *caching* genérico passou também a ser possível escolher como queremos guardar os objectos acima referidos, ou os blocos de *fragment cache*, podendo usar memória, disco ou *memcached* [33]: um sistema de *caching* de objectos, genérico, distribuído e de alta performance, bastante utilizado para melhorar a *performance* de aplicações web.

De modo a perceber qual o impacto da utilização de *memcached*, quando em conjunto com *fragment caching*, executou-se o mesmo teste da secção anterior, exactamente com os mesmos fragmentos aí descritos, mas guardando-os em *memcached* e não em disco. Os resultados são os apresentados no gráfico da Figura 28.

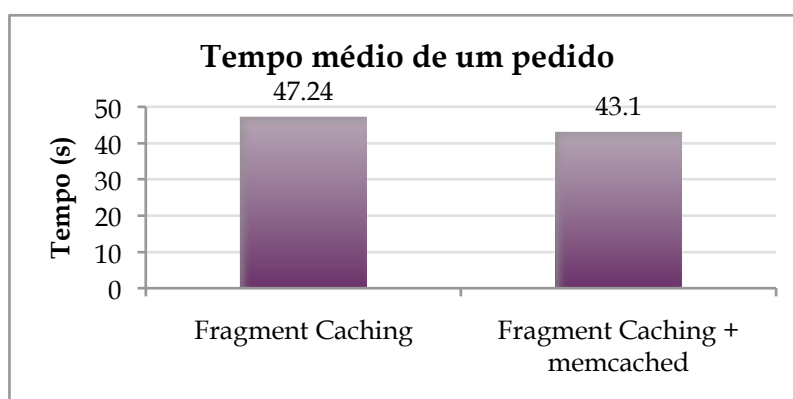


Figura 28 – Tempo médio para pedidos usando *fragment caching* e *fragment caching* com recurso a *memcached*.

De facto houve uma melhoria de performance, não muito significativa, dado que equivale apenas à diferença entre carregar uma cópia de disco ou de *memcached*. A potencialidade deste tipo de mecanismo está na sua flexibilidade. Havia outros

potenciais candidatos à utilização de *memcached* no exemplo dado, como resultados de *queries* para além dos fragmentos já guardados.

### 7.2.5. Conclusão

Como se pode ver ao longo deste capítulo, existem muitas alternativas no que toca à optimização de uma aplicação em RoR. É bastante fácil descuidar pormenores relativos à arquitectura de uma aplicação, que mais tarde podem ser fatais e implicar muito tempo de reestruturação. As estratégias aqui analisadas podem funcionar melhor ou pior consoante o caso específico da aplicação. Nenhuma aplicação funciona da mesma maneira, e a utilização, quando em ambiente de produção, é muitas vezes difícil de prever.

O mecanismo de *counter caching* pode utilizar-se com alguma segurança em todos os casos de contagem de modelos associados. A redução no número de pedidos vai compensar a desvantagem de causar redundância na base de dados. Por esta razão encoraja-se a sua utilização sempre que exista este tipo de associação uma vez que a sua implementação é simples e directa.

Sabendo-se de antemão quais os possíveis mecanismos de *caching*, em que situações se podem aplicar e quais os benefícios de cada um é possível reformular certos aspectos da aplicação para que se possa tirar partido do mais eficiente. No entanto é necessária alguma prudência para que não se utilizem estratégias de *caching* onde a expiração da cópia em *cache* seja muito frequente. Uma utilização errada de um qualquer mecanismo de *caching* pode levar por exemplo a *bottlenecks* no I/O da *cache*, obtendo-se um efeito contrário ao pretendido.

Mecanismos como *page caching* ou *action caching* não funcionam numa *view* onde existe grande dinamismo em diferentes blocos ou que envolve autenticação com apresentação de conteúdos personalizados. Por esta razão o mecanismo mais utilizado é o de *fragment caching* e na versão mais recente da *framework* apostou-se em otimizar este mecanismo com suporte a diferentes tipos de armazenamento.

O truque no planeamento da arquitectura de uma aplicação web é planejar o projecto de modo a lidar com estes problemas, desde o início. Isto não quer dizer, no entanto, que se comece logo a preparar uma aplicação para suportar biliões de linhas em tabelas da base de dados ou centenas de pedidos por segundo. A optimização prematura é mesmo desencorajada e pode ser a raiz de problemas maiores [30]. O

papel do engenheiro deve ser apenas minimizar o tempo que leva a reestruturar algum aspecto, com base num *design* cuidadoso ao longo de todo o desenvolvimento.

### 7.3. Testes

O desenvolvimento em Ruby on Rails está muitas vezes associado a um desenvolvimento ágil, não só pela natureza da própria *framework* mas também pelos requisitos de um projecto web, onde as mudanças são, normalmente, muito frequentes. Até à data, a realização de testes em Ruby on Rails não havia sido muito explorada pela empresa onde decorreu o estágio. Assim, tornou-se de interesse mútuo o estudo da melhor forma para a sua execução, tal como possíveis vantagens, atendendo ao método de trabalho em vigor. A forma de trabalhar actual tem muito em comum com *Extreme Programming* (XP). A própria topologia das instalações (em *open space*) e a entreaajuda dos vários elementos da mesma equipa estão de acordo com o ambiente aconselhado para um desenvolvimento segundo a metodologia *Extreme Programming* [34]. Para melhor compreender o funcionamento das metodologias de testes associadas a XP houve um esforço para as aplicar no desenvolvimento do *Roomblick* de modo a melhor estudar as suas vantagens e dificuldades quando aplicadas a uma aplicação real.

A *framework* Ruby on Rails está bem preparada para a realização de testes unitários e funcionais, havendo um ambiente de teste, com dados independentes, bem como um conjunto de ferramentas que ajudam na escrita e execução desses testes. Contudo, a experiência de metodologias de teste na empresa é muito reduzida. A dificuldade inicial na automatização dos testes foi apontada pelos colaboradores que trabalham com RoR como sendo o principal problema, aliado ao receio de que o tempo perdido pudesse ultrapassar o desejado, não se cumprindo as metas para finalização dos projectos. Um estudo mais aprofundado acerca de desenvolvimento ágil, *Extreme Programming* e desenvolvimento por *Test Driven Development* e *Behaviour Driven Development* pode ser consultado no Anexo C. Podem também ser consultados no Anexo L os procedimentos práticos na utilização destas duas metodologias.

### 7.3.1. Test Driven Development

*Test Driven Development* (TDD), com resultados que comprovam a sua eficácia [35], envolve uma certa sistematização e alteração no modo de planear o trabalho e isso foi a primeira dificuldade que se teve de ultrapassar. A realização de testes sem qualquer código escrito não era para o autor uma prática conhecida e inicialmente pareceu mesmo pouco natural.

Transposta essa dificuldade, existe a barreira da sintaxe utilizada nos testes, que apesar de não ser complexa é pouco intuitiva, contrastando, de certa maneira, com a filosofia da própria linguagem Ruby. A biblioteca de testes incorporada em Ruby on Rails tem uma sintaxe diferente de todo o código até então desenvolvido em Ruby, sendo uma novidade em muitos aspectos, mesmo depois de alguma experiência com a *framework* e a linguagem Ruby. Este problema é já conhecido como um entrave à utilização dos testes [36] e torna-se ainda mais complexo quando se tratam de testes funcionais onde surge a necessidade de testar *requests* AJAX ou interacção com email.

É certo que houve, de facto, alguma dificuldade, própria de um período de aprendizagem, mas que foi ultrapassada com pouco mais de um dia de experimentação. Adquiridas as noções base e algumas ajudas para a automatização, os testes unitários a um novo modelo foram feitos rapidamente e sem problemas.

### 7.3.2. Behaviour Driven Development

*Behaviour Driven Development* (BDD) é geralmente apresentado como uma evolução de TDD, onde nos devemos afastar da ideia de que estamos a escrever testes. A principal diferença para TDD é o vocabulário. Substituindo classes de teste existem contextos e em vez de se escrever métodos começados com “test” iniciam-se com “should”. A utilização deste tipo de teste em Ruby on Rails requer a instalação da *gem* ‘RSpec’ [37].

Os testes são em tudo semelhantes aos feitos com a biblioteca Ruby utilizada na metodologia TDD, mas a sintaxe é própria e muito diferente. Por esta razão, a automatização deste tipo de teste não representa menos dificuldades que utilizando a biblioteca de testes que vem por omissão com a *framework*.

A principal diferença foi sentida ao nível da manutenção. Apesar de estes testes serem escritos de forma muito mais extensa, a sua análise futura é muito mais simples e directa. Além disso, é possível exportar o resultado dos testes para

formatos como HTML (ver Figura 29), que podem funcionar como um documento de requisitos da aplicação, podendo assim ser analisado de forma simples.

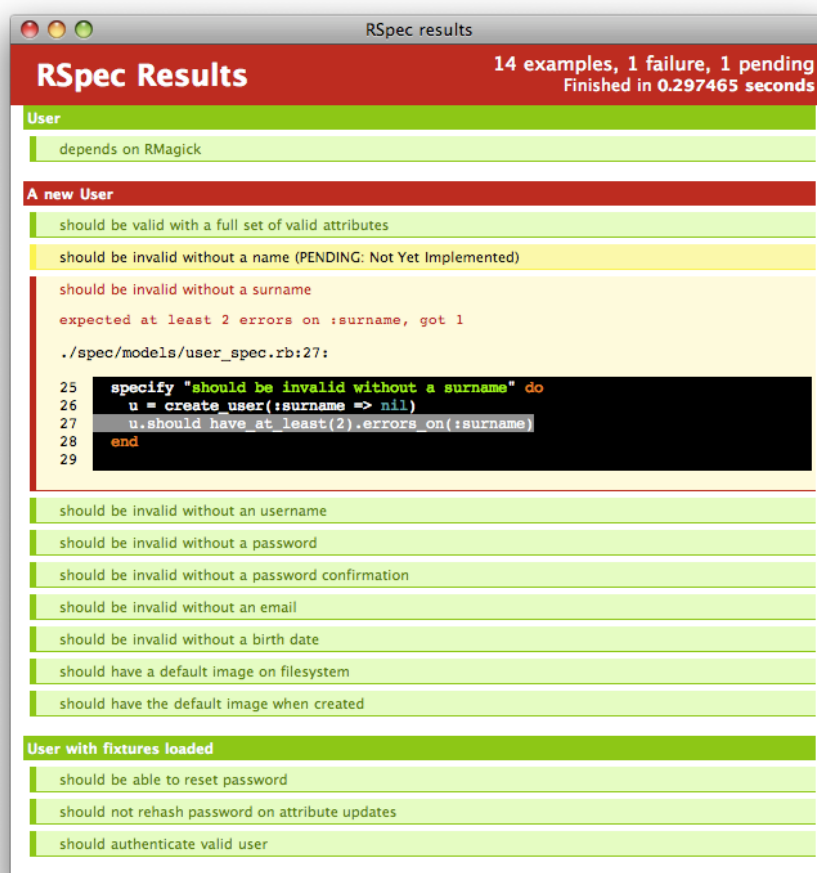


Figura 29 – Resultado de testes unitários a um modelo usando RSpec com output para um documento HTML.

Os testes que passam aparecem a verde, os que falham a vermelho com uma descrição da razão de falha e os que ainda não foram implementados a amarelo. Esta metodologia revelou-se bastante eficaz, tendo-se detectado muitos erros criados inadvertidamente durante o processo de desenvolvimento. Como já foi referido, é no entanto uma metodologia que requer muita automatização e muita habituação, e mesmo depois deste estágio não se consegue, apenas com a experiência de uma aplicação, estar completamente preparado para a realização de testes, sem falhas, logo no início de um projecto.

### 7.3.3. Conclusão

Com o crescimento em complexidade de uma aplicação, muitas vezes as funcionalidades simples implementadas numa primeira fase em que eram facilmente perceptíveis deixam de estar presentes de modo tão transparente ao longo do desenvolvimento de uma aplicação. Ao longo do desenvolvimento do *RoomBlick* não foram raros os casos em que inadvertidamente se introduziu um *bug* ou se quebrou o correcto funcionamento de uma funcionalidade implementada meses antes. Por terem sido implementadas numa fase anterior não houve posteriormente preocupação acrescida com o seu correcto funcionamento, descobrindo-se as falhas muitas vezes por mero acaso.

A escrita de testes ajuda a evitar o tipo de erro descrito. As funcionalidades melhor do que estarem documentadas, têm quase que um guião de funcionamento, que informa de algum mau comportamento que possa ocorrer. Desta forma garante-se que, ao acrescentar uma nova funcionalidade, todas as que foram até aí implementadas continuam a funcionar correctamente.

Depois de se experimentar as bibliotecas de testes descritas neste capítulo optou-se pela utilização de RSpec pelo modo de escrever e pela maior facilidade de manutenção. A única desvantagem é que os *plugins* instalados (que também trazem muitas vezes testes para os seus modelos e controladores) vêm, normalmente, apenas com testes baseados na biblioteca Ruby instalada por omissão, obrigando a um trabalho adicional. A falta de prática levou muitas vezes a fazerem-se testes apenas depois de implementado algum código uma vez que inicialmente não havia a percepção de que testes seriam necessários.

A escrita de testes é um processo demorado, uma vez que os testes (dependendo do nível de detalhe) em RSpec, por exemplo, formam ficheiros extensos (em alguns casos dez vezes mais linhas que o código que se quer testar). Além disso envolve uma prática bastante grande. Não é difícil fazerem-se testes para funcionalidades simples, mas a complexidade cresce bastante quando se trata de outras mais imbricadas ou que envolvem processos pouco comuns. Uma vez assimilado o *know-how* da escrita de testes a tarefa torna-se no entanto muito mais simples uma vez que o reaproveitamento de código é muito grande.

As dificuldades apresentadas pelos colaboradores fazem por isso todo o sentido, mas se a realização de testes fosse uma prática corrente, a longo prazo ia justificar, certamente, o tempo de aprendizagem.

## 7.4. Usabilidade

Na Web, uma boa usabilidade é uma condição necessária para a sobrevivência de uma aplicação [38]. Se uma aplicação for difícil de utilizar ou a página inicial não explicar com clareza o que se oferece e o que os utilizadores podem fazer ou se um utilizador se perde com facilidade então pode comprometer-se a viabilidade da aplicação.

É possível distinguir três níveis de estudo no que diz respeito a usabilidade [39]:

- comportamento de um utilizador nas aplicações Web em geral;
- resultados focando tipos específicos de aplicação e funcionalidades comuns;
- resultados detalhados acerca de uma aplicação particular e dos seus clientes.

Existem já bastantes publicações, resultado de anos de estudo, com milhares de utilizadores e aplicações, fornecendo uma base sólida de orientação para os dois primeiros pontos acima referidos. Na Tabela 5 apresentam-se alguns pontos em que a aplicação foi modificada pela análise de algumas linhas orientadoras [39] [40], sem que para isso tivesse de haver teste prévio.

| Parâmetro           | Modificação   |
|---------------------|---|
| Navegação           | Os menus de navegação foram separados em duas hierarquias diferentes e distintas consoante o propósito. O link para a página actual foi removido para evitar confusão e distinguido dos restantes.      |
| Internacionalização | A selecção do idioma passou a apresentar também o nome (ao invés de ser apresentado apenas por icons com as bandeiras) e aparece em toda a aplicação em vez de obrigar à selecção na página de entrada. |
| Internacionalização | O idioma que é apresentado por omissão deve ser fruto da comunicação directa entre o servidor e o cliente, de modo a acertar-se qual o mais indicado, sem que o utilizador tenha de intervir.           |
| Preços              | Os preços foram formatados para um tamanho maior, bem visível aparecendo o mais cedo possível (na listagem de anúncios).  |
| Página de entrada   | A página de entrada foi reformulada de modo a ter cabeçalhos bem distintos e transmitir de modo conciso qual o objectivo da aplicação.  |

*Tabela 5 – Lista de alterações relacionadas com usabilidade feitas numa primeira versão da aplicação RoomBlick.*



Para obter resultados relativos ao terceiro ponto é necessário, no entanto, efectuar testes de usabilidade para a aplicação desenvolvida, uma vez que só assim se obtêm resultados específicos, de grande valor para o sucesso da aplicação.

#### **7.4.1. Testes de usabilidade**

Um dos principais problemas que sobressai da análise das aplicações semelhantes à desenvolvida neste estágio é, de resto, o pouco cuidado com a usabilidade. As linhas orientadoras são claramente ignoradas em muitos aspectos (ver Anexo F) o que acentua ainda mais a necessidade de ser desenvolvida uma aplicação forte neste ponto, já por si decisivo.

O método de teste utilizado é o descrito por Jakob Nielsen [38], consultor de usabilidade web conceituado, que pode ser chamado de “pensar alto”. Os utilizadores têm um guião com tarefas chave que se quer testar e é pedido que vão expressando verbalmente aquilo em que estão a pensar. É bom saber que um utilizador não carrega no botão certo para uma determinada acção, mas é muito mais valioso saber porquê.

A metodologia aconselhada seria a realização de testes iterativamente, começando-se desde o início com testes utilizando protótipos em papel, mesmo antes de qualquer implementação, seguindo-se testes com a aplicação sempre que houvesse dúvidas acerca da usabilidade de determinadas funcionalidades.

Não foi, no entanto a metodologia seguida no desenvolvimento do *RoomBlick*, uma vez que quando foi estudado o processo de testes de usabilidade já existia uma versão funcional da aplicação. Por essa razão foram feitas as alterações descritas na Tabela 5. Só depois foi feita uma iteração de testes que serve de exemplo neste documento, mas não deverá ser única, havendo a necessidade de, após feitas as alterações necessárias (fruto da análise dos resultados destes testes), proceder-se a novos testes.

No Anexo M pode consultar-se o guião de testes utilizado na sessão realizada no passo intermédio acima descrito. Foram considerados alguns aspectos específicos para os quais se pretendia estudar o comportamento dos utilizadores, como por exemplo o sistema de *drag and drop* na reordenação de imagens na edição de um anúncio, bem como tentar detectar possíveis erros que não tivessem sido tomados como tal ou possíveis áreas de melhoramento. Desta forma, e como a aplicação

apresentava já um nível de complexidade bastante elevado, o guião tenta cobrir as operações principais. Tudo o que é relativo à usabilidade do sistema de administração terá de ser testado à parte uma vez que o público alvo é diferente (provavelmente pessoas na empresa).

Os testes foram realizados a 8 utilizadores (geralmente 5 a 8 utilizadores são suficientes, dependendo da complexidade do sistema, para detectar fontes de problemas de usabilidade [38]). Os testes de usabilidade envolvem a medição de quão bem os utilizadores respondem às quatro seguintes áreas:

- Eficácia – quanto tempo é gasto no cumprimento de cada tarefa?
- Direcção – quantos erros são dados? Esses erros são fatais ou recuperáveis?
- Memorização – quanto é que a pessoa se recorda depois de algum tempo de não utilização?
- Resposta emocional – como é que a pessoa se sentiu ao completar as tarefas? Confiante, stressado? Recomendaria este sistema a um amigo?

Os resultados da primeira iteração de testes, que podem ser consultados no devem ser então usada como base para comparativo com testes subsequentes. Alguns aspectos da aplicação que apresentaram problemas de usabilidade nos testes realizados serão fruto de análise futura, mas outros foram já alterados por haver bastante confiança no seu impacto negativo e serem de resolução relativamente fácil. Um exemplo disso é a listagem de comodidades de um quarto. Quando foi pedido que utilizadores identificassem se determinado quarto dava acesso à utilização de frigorífico houve muitas dificuldades em completar a tarefa, uma vez que a listagem dessa informação era feita com omissão das comodidades a que o quarto não dava acesso. Um exemplo desse tipo de listagem está apresentado na Figura 30.



*Figura 30 – Apresentação das comodidades de um quarto em lista sequencial omitindo as que não são facultadas.*

Isto implica que, não havendo Internet, esse parâmetro não figure na listagem. Os utilizadores que não sabem da sua existência não se apercebem assim de quais as comodidades em falta. A solução encontrada, que foi também sugerida por dois utilizadores foi a listagem de todas as comodidades possíveis indicando quais as que se tem acesso como está representado no protótipo da Figura 31.



Figura 31 – Alternativa à apresentação das comodidades, apresentando uma lista com todas as possíveis.

#### 7.4.2. Conclusão

Os utilizadores de aplicações web são, por natureza, impacientes e não gostam de perder tempo na tentativa de perceber como fazer algumas operações. Abandonar a aplicação é a defesa mais comum a este tipo de dificuldades, procurando-se alternativas de outras empresas. Os testes de usabilidade são a técnica usada para garantir que os utilizadores conseguem efectuar as tarefas eficientemente e com satisfação.

Os testes realizados sobre a aplicação *RoomBlick* comprovaram a sua necessidade no sentido em que muitos problemas encontrados nunca teriam sido encarados como tal. Os resultados devem ser interpretados não só tendo em conta o número de utilizadores que apresentaram as mesmas dificuldades mas também analisando qual o impacto que esses problemas têm na interacção do utilizador com a aplicação, qual o esforço de implementação de uma solução e qual a gravidade dos erros possivelmente cometidos.

## 8. Conclusões

---

De uma forma geral, os objectivos inicialmente propostos foram cumpridos e considera-se que o estágio foi bem sucedido. Foi depositada confiança nos resultados desenvolvidos pelo estagiário, pelas suas capacidades técnicas e organizacionais demonstradas.

Ao longo de todo o trabalho houve um esforço por não se focar demasiado a aplicação desenvolvida, que deve ser vista como um exemplo prático de teste de funcionalidades e metodologias gerais, a aplicar noutras aplicações futuras. No final da realização do estágio, mais do que a familiarização com o ambiente empresarial ou desenvolvimento de uma aplicação, deixou-se um contributo em termos de *know-how* no desenvolvimento em Ruby on Rails e na integração de diversas tecnologias que até aí não haviam sido exploradas.

Foi feita também uma actualização constante dos conhecimentos adquiridos inicialmente, uma vez que a *framework* evoluiu bastante durante o tempo de estágio e o estagiário estava em melhor posição para absorver e partilhar as últimas evoluções do que os colaboradores da empresa que trabalham com prazos estabelecidos em aplicações estáveis e não tiveram oportunidade de as estudar e experimentar.

Como foi analisado no capítulo 4 a migração para RoR foi uma estratégia válida, no contexto da empresa, permitindo um desenvolvimento de aplicações web de forma muito mais rápida. No entanto esta migração foi feita recentemente e havia alguns aspectos relacionados com a *framework* e até com o desenvolvimento web em geral que se queriam ver esclarecidos mas para os quais ainda não tinha sido encontrada resposta. Aspectos como a internacionalização, mecanismos de *caching*, testes da aplicação e usabilidade foram considerados muito interessantes para a empresa.

Com esta experiência, além de ter havido a aquisição de conhecimento no desenvolvimento de aplicações com RoR, o estagiário adquiriu ainda noções fundamentais para desenvolvimento de aplicações web que até aí eram completamente desconhecidas.

### 8.1.1. Trabalho futuro

A aplicação desenvolvida ainda não foi finalizada tendo havido um convite para que o trabalho fosse continuado até uma versão final. Assim, existe à partida um conjunto de tarefas a serem concluídas ou iniciadas, entre as quais se destaca:

- novas iterações de testes de usabilidade;
- desenvolvimento de *plugins* para algumas funcionalidades de modo a serem partilhadas noutros projectos;
- finalização da internacionalização da aplicação;
- incluir um sistema de comparação directa de quartos seleccionados;

## 9. Referências

---

1. <http://www.redmine.org/> - Sistema de gestão de projectos Redmine.
2. <http://rails100.pbwiki.com/> - Lista de aplicações web desenvolvidas em Rails.
3. <http://oodt.jpl.nasa.gov/better-web-app.mov> - Vídeo comparativo de frameworks de desenvolvimento de aplicações web.
4. <http://www.mysql.com/> - Página da base de dados *opensource* MySQL.
5. <http://www.djangoproject.com/> - Página oficial da *framework* Django.
6. <http://www.zope.org/> - Página oficial da *framework* Zope.
7. <http://plone.org/> - Página oficial da *framework* Plone.
8. <http://cakephp.org/> - Página oficial da *framework* CakePHP.
9. <http://java.sun.com/javaee/> - Página oficial do Java EE.
10. <http://www.springframework.org/> - Spring Framework.
11. <http://www.hibernate.org/> - Página oficial do Hibernate.
12. <http://tomcat.apache.org/> - Apache Tomcat.
13. <http://www.eclipse.org/> - Página oficial do Eclipse IDE
14. <http://www.w3.org/> - World Wide Web consortium.
15. <http://validator.w3.org/> - Ferramenta de validação de documentos web.
16. <http://www.marketwire.com/mw/release.do?id=818650> - Artigo: "Technology Leaders Join OpenID Foundation to Promote Open Identity Management on the Web".
17. <http://openid.net/get/> - Lista de facilitadores de serviço OpenID.

18. <http://www.openidenabled.com/ruby-openid/> - Biblioteca de interacção com OpenID para Ruby.
19. <http://identity.eastmedia.com/identity/show/Consumer+Plugin/> - *Plugin* para autenticação com OpenID desenvolvido pela EastMedia.
20. [http://dev.rubyonrails.org/browser/plugins/open\\_id\\_authentication/](http://dev.rubyonrails.org/browser/plugins/open_id_authentication/) - *Plugin* para autenticação com OpenID desenvolvido pela equipa do Ruby on Rails.
21. <http://www.poiportugal.com/> - Listagem de pontos de interesse em Portugal.
22. <http://www.gpsbabel.org/> - Página oficial do GPSbabel - conversor de pontos de interesse.
23. <http://maps.google.com/> - Google Maps.
24. [http://reviews.cnet.com/4520-9239\\_7-6526615-3.html](http://reviews.cnet.com/4520-9239_7-6526615-3.html) - Comparativo de sistemas de representação geográfica.
25. <http://geokit.rubyforge.org/> - *Plugin* para georreferenciação.
26. <http://earth.google.com/> - Página oficial da aplicação Google Earth.
27. <http://agilewebdevelopment.com/plugins/category/8/> - Lista de *plugins* para internacionalização da *framework* Ruby on Rails.
28. <http://www.gnu.org.ua/software/gettext/> - Página oficial do GNU Gettext.
29. <http://www.poedit.net/> - Página oficial do programa Poedit.
30. Henderson, C., *"Building Scalable Web Sites"*, O'Reilly Media, 2006
31. <http://www.modrails.com/> - Página oficial do servidor Phusion Passenger.
32. <http://httpd.apache.org/> - Página oficial do servidor web Apache.
33. <http://www.danga.com/memcached/> - Página oficial do *memcached*.
34. Ron Jeffries, Ann Anderson, Chet Hendrickson, *"Extreme Programming Installed"*, Addison Wesley Longman, 2001.

35. Michael, E.; Williams, L., "Assessing Test-Driven Development at IBM", Proc. 25th Int'l Conf. Software Eng (ICSE 03), IEEE CS Press, 2003, pp 564-569.
36. Astels, Dave, "A New Look at Test-Driven Development" - Artigo disponível em [http://blog.daveastels.com/files/BDD\\_Intro.pdf](http://blog.daveastels.com/files/BDD_Intro.pdf).
37. <http://rspec.info/> - Página da gem RSpec com documentação e downloads.
38. Nielsen, J., "Usability 101: Introduction to Usability" - Artigo disponível em <http://www.useit.com/alertbox/20030825.html>.
39. Nielsen, J.; Loranger, H., "Prioritizing Web Usability", New Riders, 1st Edition, 2006.
40. Nielsen, J., "Top 10 Application-Design Mistakes" - Artigo disponível em <http://www.useit.com/alertbox/application-mistakes.html>.
41. Benk, Kent, "eXtreme Programming explained : embrace change", Addison-Wesley Professional, 1999.
42. Marshal, Kevin, "Web Services on Rails", O'Reilly, 2006.
43. Bradburne, Alen, "Practical Rails Social Networking Sites", Apress, 2007.
44. Thomas, Dave; Hansson, David, "Agile Web Development with Rails - Second Edition", Pragmatic Bookshelf, 2006.
45. Thomas, Dave, "Programming Ruby: The Pragmatic Programmer's Guide, Second Edition", Pragmatic Bookshelf, 2004.
46. <http://www.rubyonrails.org/> - Página oficial da framework Ruby on Rails.
47. <http://java.sun.com/blueprints/patterns/MVC-detailed.html> - Arquitetura MVC explicada em detalhe.
48. <http://www.prototypejs.org/> - Página oficial da biblioteca Prototype.
49. <http://www.script.aculo.us/> - Página oficial da biblioteca Script.aculo.us.
50. <http://rmagick.rubyforge.org/> - Página oficial da gem RMagick.



# Anexos

# Índice

---

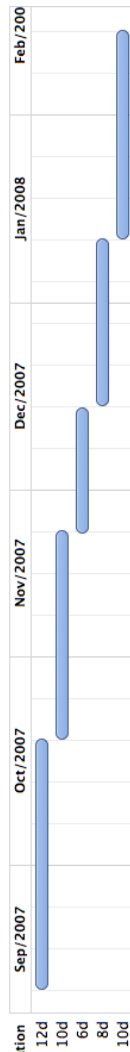
|   |    |
|---|----|
| Anexo A - Planeamento .....                                 | 3  |
| Anexo B - Framework Ruby on Rails .....                     | 4  |
| 1. Origem e filosofias.....                                 | 4  |
| 2. Model-View-Controller .....                              | 4  |
| 3. Scaffolding .....  | 7  |
| 4. Gems e Plugins .....                                     | 7  |
| 5. Templates e Helpers .....                                | 8  |
| 6. AJAX.....  | 9  |
| 7. Migrações.....   | 10 |
| 8. Web services.....  | 11 |
| 9. Servidores HTTP .....                                    | 12 |
| 10. Caching.....  | 13 |
| 11. Deployment.....   | 17 |
| Anexo C - Desenvolvimento ágil.....                         | 18 |
| 1. Filosofias .....   | 18 |
| 2. Extreme Programming.....                                 | 19 |
| 3. Test Driven Development (TDD) .....                      | 21 |
| 4. Behaviour Driven Development (BDD) .....                 | 22 |
| 5. TDD e BDD em Rails .....                                 | 23 |
| Anexo D - Especificação da aplicação de estudo .....        | 25 |
| 1. Requisitos .....   | 25 |
| 2. Diagramas de casos de uso .....                          | 26 |
| 3. Modelo de dados .....                                    | 27 |
| 4. Protótipo.....   | 27 |
| Anexo E - Especificação da aplicação RoomBlick .....        | 30 |
| 1. Requisitos .....   | 30 |
| 2. Diagramas de casos de uso .....                          | 33 |
| 3. Protótipo.....   | 35 |
| Anexo F - Estado da arte em aplicações de arrendamento..... | 43 |

|   |    |
|---|----|
| 1. Introdução .....                                       | 43 |
| 2. Bquarto .....  | 43 |
| 3. Easyquarto.....  | 46 |
| 4. Conclusões .....                                       | 49 |
| Anexo G - Integração com OpenID.....                      | 50 |
| Anexo H - Implementação de georreferenciação.....         | 53 |
| Anexo I - Guia de utilização de Gettext Localization..... | 58 |
| Anexo J - Pesquisa .....                                  | 61 |
| 1. Questionário.....                                      | 61 |
| 2. Resultados .....                                       | 62 |
| 3. Implementação .....                                    | 63 |
| Anexo K - Guia de implementação de caching.....           | 65 |
| 1. Counter caching.....                                   | 65 |
| 2. Page caching e Action caching.....                     | 66 |
| 3. Sweepers .....   | 66 |
| 4. Fragment caching .....                                 | 67 |
| Anexo L - Guia de realização de testes .....              | 69 |
| 1. Test Unit.....   | 69 |
| 2. Rspec.....   | 71 |
| 3. Fixtures .....   | 73 |
| Anexo M - Usabilidade.....                                | 74 |
| 1. Questionário.....                                      | 74 |
| 2. Resultados .....                                       | 75 |

# Anexo A

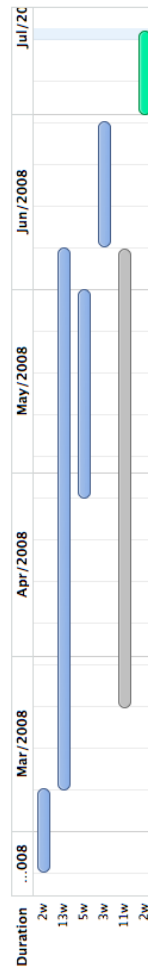
## Planeamento

### Primeira Fase



- Task
- 1) Estudo de Ruby on Rails e...
  - 2) Estudo de outras frameworks
  - 3) Levantamento de requisitos
  - 4) Implementação módulo slideshow
  - 5) Implementação módulo editor

### Segunda Fase



- Task
- 1) Migração do trabalho feito na primeira fase
  - 2) Implementação
  - 3) Testes de usabilidade
  - 4) Estudo comparativo de mecanismos de caching
  - 5) Estudo e realização de testes
  - 6) Elaboração do relatório

Figura 1 – Diagramas de Gantt de planeamento das duas fases do projecto

# Anexo B

---

## Framework Ruby on Rails

### 1. Origem e filosofias

Ruby on Rails (Rails) é uma *framework* desenvolvida em Ruby – uma linguagem de programação não compilada, orientada a objectos, com uma sintaxe limpa – com vista à implementação de aplicações web com recurso a uma base de dados. Foi inventada por David Heinemeier Hansson (1979 – Copenhaga), programador dinamarquês, tendo sido utilizada numa aplicação que estava a desenvolver – Basecamp ([www.basecamp.com](http://www.basecamp.com)) – e disponível ao público em Julho de 2004.

Dando prioridade ao modo simples de fazer as coisas e tirando partido da linguagem Ruby permite uma alta produtividade no desenvolvimento de aplicações web. Meses após o lançamento oficial esta *framework* passou de desconhecida a uma das *frameworks* de escolha para implementação de uma variedade das chamadas aplicações Web 2.0, não sendo apenas uma moda junto dos mais entusiastas mas também adoptada por multinacionais para desenvolvimento de aplicações web robustas.

Uma aplicação em Rails usa convenções simples de programação que permitem um mínimo de configuração e segue as seguintes filosofias chave:

- **Don't repeat yourself (DRY):** reduzir duplicações, particularmente quando se trata de programação. A informação está localizada num único sítio não ambíguo.
- **Convention over configuration (CoC):** significa que apenas temos de especificar aspectos não convencionais da nossa aplicação.

### 2. Model-View-Controller

Em aplicações complexas, que lidam com um volume elevado de dados, pretende-se, normalmente, separar os dados e a interface gráfica para que as alterações à interface

não afectem os dados e estes possam ser reorganizados sem interferências com a primeira.

A arquitectura MVC (ver Figura 2) resolve este problema separando o acesso da apresentação dos dados bem como a interacção com o utilizador, introduzindo uma componente intermédia: o *controller* (controlador). Com a decomposição em *models* (dados) e *views* (interface), a arquitectura MVC ajuda a reduzir a complexidade no *design* da arquitectura das aplicações e aumenta a flexibilidade e reutilização.

As componentes da arquitectura MVC são então os seguintes:

- **Model:** representação da informação, dependente do domínio sobre a qual a aplicação opera. Numa arquitectura MVC o método de armazenamento e de acesso aos dados não é especificamente mencionado uma vez que está num nível inferior e encapsulado pelo Modelo.
- **View:** transforma o modelo numa forma passível de interacção, tipicamente um elemento de interface com o utilizador. Podem existir várias views para o mesmo modelo com propósitos distintos.
- **Controller:** processa e responde a eventos, tipicamente acções do utilizador, e pode invocar mudanças no modelo.

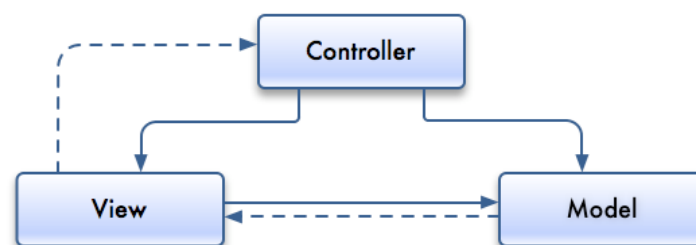


Figura 2 – Arquitectura MVC (relações directas a cheio e indirectas a tracejado).

O MVC é muitas vezes visto em aplicações web, onde as *views* são o código HTML da página actual, o *controller* é o código que recolhe dados dinâmicos e gera o conteúdo do HTML e o *model* é representado pelo conteúdo, normalmente guardado numa base de dados ou ficheiros XML.

Na Figura 3 está esquematizada a implementação MVC na *framework* Ruby on Rails. O Ruby on Rails implementa este tipo de arquitectura onde o *model* fica ao cargo de uma camada de ORM chamada *Active Record*. O *Active Record* permite definir

relações entre diferentes tabelas de dados de forma intuitiva (ex: *has\_many*, *belongs\_to*), e faz validações nos modelos.

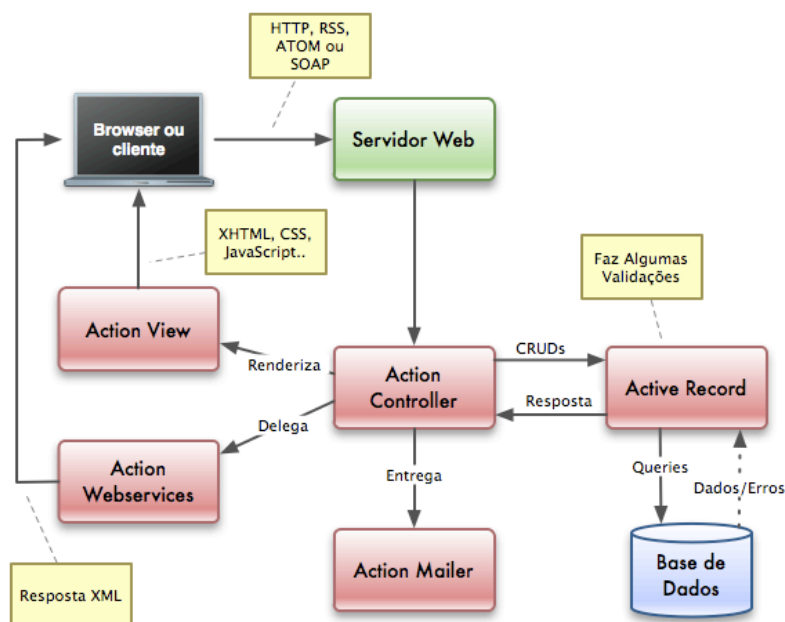


Figura 3 – Arquitectura MVC na framework Ruby on Rails.

Esta camada permite a apresentação das linhas de uma base de dados como objectos e enriquece estes objectos com métodos da lógica de negócio. O *controller* e a *view* são manipulados pelo *Action Pack*, que é constituído por duas partes diferentes: *Action View* e *Action Controller*.

O *Action Controller* recebe os pedidos do servidor web, faz os pedidos necessários ao *Active Record* e devolve os resultados. É também responsável pelas sessões e *cookies*, cria todos os URL's da aplicação automaticamente consoante o nome dos controladores e as suas acções (geralmente apresentado no seguinte formato: `http://endereço/controlador/acção/parâmetros`).

O *Active View* devolve o resultado do pedido numa view em XHTML, XML, imagens, Javascript, etc. Normalmente cada acção de um controlador tem uma view e cada controlador tem uma *view* especial chamada *layout* que se renderiza sempre, independentemente da acção que é chamada.

Pode ainda ver-se na Figura 3 que existem outras duas componentes diferentes – *Action Webservice* e *Action Mailer*. A primeira implementa o suporte para *web services* com protocolos SOAP e XML-RPC e permite declarar e publicar API's de modo

simples. O *Action Mailer* permite interagir directamente com utilizadores ou outros sistemas por email.

### 3. Scaffolding

Um *scaffold* em Rails é uma estrutura gerada automaticamente para manipulação de um modelo. Dando uso a todas as convenções presentes na *framework* pode desta forma criar-se interfaces básicas de administração utilizando apenas um comando. Pela análise da base de dados são construídas interfaces para operações CRUD sobre um objecto pertencente ao modelo correspondente.

Os *scaffolds* não devem no entanto ser usados para criar aplicações ou partes integrantes destas, uma vez que, normalmente, as aplicações necessitam de uma interface mais optimizada e muitas vezes com particularidades específicas. Podem (e devem) por outro lado ser usados como ponto de partida, sobre o qual se pode construir a aplicação, fazendo as alterações necessárias.

### 4. Gems e Plugins

Uma *gem* é uma aplicação em Ruby ou uma biblioteca particular, desenvolvida por outra pessoa e disponível num repositório *on-line*. Usando uma única linha de comandos pode-se instalar, remover ou actualizar as aplicações ou bibliotecas instaladas, tendo-se desta forma total controlo sobre o estado actual da plataforma Ruby e extensões na nossa máquina.

Por exemplo para instalar a biblioteca que faz com que o resultado de testes apareça a verde ou vermelho consoante o sucesso ou falha basta escrever

```
hobbes:~ ricardo$ sudo gem install redgreen
```

obtendo-se o seguinte *output*:

```
Updating metadata for 129 gems from http://gems.rubyforge.org
.....
complete
Successfully installed redgreen-1.2.2
1 gem installed
```



Os *plugins* em Rails são uma extensão ou modificação da *framework*, com código desenvolvido por terceiros. A utilização de *plugins* nas aplicações é bastante comum, não só por ser extremamente fácil de as integrar mas porque normalmente são suficientemente flexíveis e evitam o desenvolvimento de determinadas funcionalidades de raiz.

O funcionamento é em tudo semelhante ao das *gems* mas neste caso as funcionalidades ficam apenas afectas à nossa aplicação e não a toda a plataforma Ruby.

De seguida mostra-se o *output* na instalação de um *plugin* que permite tratar um modelo como uma árvore binária - *acts\_as\_tree*, a título de exemplo. Todas as aplicações têm um gestor de *plugins* incorporado por omissão em *./script/plugin*.

```
hobbes:railsapp ricardo$ script/plugin install acts_as_tree
+ ./README
+ ./Rakefile
+ ./init.rb
+ ./lib/active_record/acts/tree.rb
+ ./railsapp/abstract_unit.rb
+ ./railsapp /acts_as_tree_test.rb
+ ./railsapp /database.yml
+ ./railsapp /fixtures/mixin.rb
+ ./railsapp /fixtures/mixins.yml
+ ./railsapp /schema.rb
```

## 5. Templates e Helpers

Em Ruby on Rails, quando criamos uma *view* estamos a criar um *template* - algo que vai ser tratado para gerar um resultado final. Os templates contêm assim uma mistura de texto e código, usado para permitir conteúdo dinâmico. Este código embutido permite ter-se acesso à informação constante num controlador. A *framework* vem com suporte para três diferentes tipos de template:

- **RXML** - templates usados para construir respostas em XML;
- **RHTML** - templates constituídos por uma mistura de HTML e Ruby embutido, usados tipicamente para gerar páginas HTML;

- **RJS** - templates que geram código Javascript executado no browser e são geralmente usados para interacção com AJAX.

O modo como se constroem estas *views* pode variar, havendo um grande leque de sistemas de template que se podem utilizar. Por omissão usa-se o ERB, que faz parte da biblioteca standard do Ruby e usa *tags* especiais dentro das quais se escreve Ruby directamente.

Os *helpers* são módulos que fornecem métodos automaticamente disponíveis para utilização nos templates e que geram HTML (ou XML, ou Javascript) estendendo assim o comportamento de um *template*. Desta maneira simplificam-se as *views* mantendo-se blocos de programação em Ruby em ficheiros separados. Ao mesmo tempo evitam-se duplicações em diferentes templates uma vez que o código dos *helpers* pode ser reutilizado.

A *framework* vem com uma grande colecção de *helpers* e podem ainda ser complementados com *helpers* personalizados para cada aplicação. A vasta diversidade de *helpers* usados em Ruby on Rails é de tal forma útil que foram implementados métodos semelhantes em Python para uso nas diferentes *frameworks* desenvolvidas nesta linguagem.

## 6. AJAX

AJAX (*Asynchronous Javascript And XML*) não é uma tecnologia mas sim um conjunto de várias tecnologias usadas em conjunto permitindo novas funcionalidades. AJAX incorpora:

- apresentação baseada em standards usando XHTML e CSS;
- interacção dinâmica usando DOM (*Document Object Model*);
- interacção e manipulação de dados usando XML e XSLT;
- pedidos assíncronos de dados (usando XMLHttpRequest);
- Javascript ligando todos os pontos anteriores.

O XMLHttpRequest é um objecto Javascript criado pela Microsoft, fazendo hoje parte da especificação DOM e torna possível a comunicação assíncrona com o servidor. Na realidade nem sequer é necessário usar XML e a parte verdadeiramente importante é que todos os pedidos são assíncronos.

O Ruby on Rails usa uma biblioteca Javascript - Prototype que permite abstracção completa do browser cliente e ao mesmo tempo simplifica todo um conjunto de operações comuns como validações e manipulação de objectos DOM. Com Prototype pode também usar-se pedidos AJAX de forma simples.

Para efeitos de interacção dinâmicos a *framework* vem também com outra biblioteca específica para esse fim - Script.aculo.us que não é mais que um conjunto de objectos Javascript implementados sobre Prototype, podendo por isso aproveitar todas as potencialidades de AJAX.

Todas as funcionalidades de Prototype, bem como os efeitos disponibilizados pelo Script.aculo.us estão disponíveis directamente no Ruby on Rails, sem haver necessidade de escrever código Javascript, usando-se templates RJS como foi referido na secção anterior.

## 7. Migrações

A *framework* Ruby on Rails tem um sistema de migrações, que ajuda a manter os dados sincronizados com o código actual, funcionando como um sistema de controlo de versões dos dados, não só entre diferentes pessoas numa equipa mas também quando a aplicação é transferida para o servidor em produção. Acrescentando ficheiros com as alterações desejadas e correndo um único comando assegura-se que a estrutura de dados utilizada é da versão desejada. Um exemplo de migração para a criação de um modelo *Image* pode ser:

```
class CreateImages < ActiveRecord::Migration
  def self.up
    create_table :images do |t|
      t.integer :ad_id
      t.string :name
      t.integer :num
    end
  end

  def self.down
    drop_table :images
  end
end
```

o método `self.up` corre a migração anterior, enquanto que o `self.down` permite voltar atrás caso se migre para uma versão anterior dos dados.

Tal como acontece com Javascript, no desenvolvimento de uma aplicação em Rails raramente será necessário escrever SQL. Isto aplica-se inclusive à definição e alteração da estrutura de dados através do sistema de migrações. Mantendo a definição da estrutura de dados em ficheiros Ruby, sem recurso a SQL, assegura-se que esta é agnóstica aumentando a portabilidade.

## 8. Web services

O *Action Webservice* permite o suporte de protocolos SOAP e XML-RPC nas aplicações Rails. Converte as invocações remotas de métodos em chamadas a métodos no nosso *web service* e trata da resposta. Assim, deixa-nos concentrar no trabalho de escrever os métodos específicos da nossa aplicação para servir os pedidos. Pode-se assim, não só permitir definir uma API da nossa aplicação, mas também adicionar suporte para funcionalidades disponibilizadas por outras aplicações.

Além disso a *framework* inclui um bom suporte para REST. O termo REST foi introduzido por um dos principais autores do protocolo HTTP no ano 2000 e tem vindo a ser usado desde então. Este nome refere-se a um conjunto de princípios de arquitectura de redes que definem como é que os recursos são definidos e endereçados, mas é muitas vezes usado num sentido mais lato para descrever qualquer interface simples que transmite dados específicos de uma aplicação através de HTTP sem uma camada adicional como SOAP ou HTTP *cookies* onde os verbos usado no protocolo HTTP (*GET, POST, PUT, DELETE*) devem ser usados com um significado.

A facilidade de utilização de XML é importante no que respeita a integração de sistemas. Os computadores necessitam frequentemente de trocar informação entre sistemas incompatíveis. XML é independente de formatos de transporte e de armazenamento, legível por humanos e pode ser editado em qualquer editor de texto. Gerar respostas formatadas em XML é bastante fácil, como se pode ver pelo exemplo seguinte:

```
class HatsController < ApplicationController
```

```

...
def hatsml
  @hats = Hats.find(:all)
  render :layout => false
end
. . .
end

```

podemos criar uma vista que não gera HTML como normalmente acontece, mas sim XML. A *framework* Rails está bastante bem preparada para se conseguir construir esse XML de forma fácil e elegante:

```

xml.instruct! :xml, :version=>"1.0"
xml.channel{
  for hat in @hats
    xml.hat do
      xml.title(hat.name)
      xml.description(hat.description)
    end
  end
}

```

A *framework* tem um bom suporte para XML básico, mas carece de um bom *parser* para XML envolvendo *namespaces*, *entities* etc. Este problema não tem directamente a ver com Ruby on Rails mas sim com as bibliotecas existentes até à data em Ruby e essa falha tem sido alvo de críticas.

## 9. Servidores HTTP

Para testar e correr uma aplicação em Rails é necessária a familiarização com, pelo menos, alguns dos possíveis servidores HTTP:

- **Apache** - apesar do Mongrel ser excelente como servidor para Rails, quando se fala em configuração, velocidade e estabilidade Apache bate todos os concorrentes. É também o servidor com maior difusão na Internet. Interessa principalmente por poder fazer de front-end de um cluster de servidores Mongrel para onde são distribuídos os vários pedidos.

- **nginx** - um servidor relativamente recente com bastante crescimento nos últimos anos e que serve, hoje em dia cerca de 4% das aplicações na Internet. Tal como o Apache faz de distribuidor de pedidos com bastante eficiência e pode ser utilizado como alternativa.
- **WEBrick** - para simplificar o desenvolvimento e teste rápido, quando se cria uma nova aplicação tem-se ao dispor este servidor HTTP, simples e leve. Não é, no entanto, aconselhável o seu uso em aplicações em produção, por não ser eficiente nem seguro.
- **Mongrel** - servidor HTTP com suporte para Ruby on Rails desenhado para ser rápido, leve e muito seguro.
- **Phusion Passenger** - lançado recentemente (Abril 2008) é o ambiente de produção de instalação mais fácil uma vez que é apenas necessária a instalação de uma *gem*. Passenger funciona como um módulo de Apache, à semelhança do módulo existente para PHP.
- **Thin** - tal como o anterior, também este servidor prima pela facilidade de instalação, mas pode funcionar de forma independente, não necessitando de nenhum *software* adicional.

## 10. Caching

Muitas aplicações passam muito tempo a fazer as mesmas operações inúmeras vezes. Um *blog* apresenta a lista de artigos actuais para cada visitante. Uma loja *on-line* mostra a mesma página de produtos para todos os utilizadores que a consultem.

Toda esta repetição custa recursos e tempo no servidor. O *render* de uma página de um *blog* pode envolver meia dúzia de pedidos à base de dados, correndo ao mesmo tempo um conjunto de métodos Ruby e *templates*. Não é problemático para um pedido individual, mas multiplicando por centenas ou milhares de pedidos numa hora pode atrasar os tempos de resposta. Em situações como esta, pode-se recorrer ao *caching* para reduzir significativamente a carga no servidor e aumentar a velocidade de resposta da aplicação.

Em vez de se gerar o mesmo conteúdo do zero, consecutivamente, é gerado apenas uma vez e guardado o resultado. No próximo pedido ao mesmo conteúdo devolve-se o conteúdo da *cache* em vez de se gerar novamente.

Em Ruby on Rails temos três formas de fazer *cacheing*:

1. *Page caching* – guarda em *cache* uma *view* completa.
2. *Action caching* – guarda em *cache* uma acção completa num controlador.
3. *Fragment caching* – guarda em *cache* fragmentos específicos numa *view*.

*Page caching* é a forma mais simples e mais eficiente de fazer *cache*. Da primeira vez que o utilizador faz *request* a um determinado URL, a nossa aplicação é invocada e gera o HTML de *output* como indicado na Figura 4.

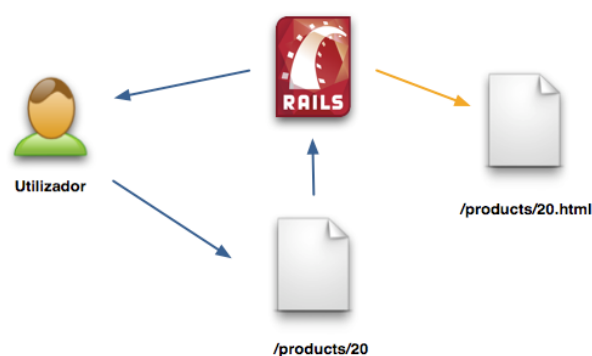


Figura 4 – Mecanismo de *page caching* no caso de não haver ainda o ficheiro HTML no disco (*cache miss*).

Os conteúdos desta página são então guardados no disco, num ficheiro HTML. No próximo *request* ao mesmo URL o HTML correspondente é servido directamente. A aplicação, na verdade, nem chega a tratar este segundo *request*, não envolvendo, por completo, interacção com Rails, como é visível na Figura 5. O *request* é tratado directamente pelo servidor, o que torna este tipo de *cacheing* muito eficiente.

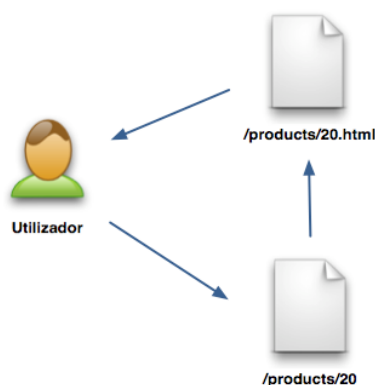


Figura 5 – Mecanismo de *page caching*, servindo um ficheiro HTML guardado previamente (*cache hit*).

No entanto, em aplicações dinâmicas, muitas vezes não é possível apresentar o conteúdo directamente de uma cópia guardada previamente. Por exemplo, se a página em questão estiver dependente de uma autenticação. Nestes casos, poderá ser usado *Action Caching*, que é em tudo semelhante ao *Page Caching* mas que tem sempre de aceder a código da nossa aplicação, correndo os filtros para acções que assim o requeiram. Assim, num primeiro pedido o funcionamento é semelhante ao descrito na Figura 5, mas em pedidos posteriores à mesma página recorre sempre à aplicação em Rails como se pode ver na Figura 6. É por isso consideravelmente mais lento que o primeiro, mas, pode ainda assim reduzir drasticamente o tempo de resposta de um pedido.

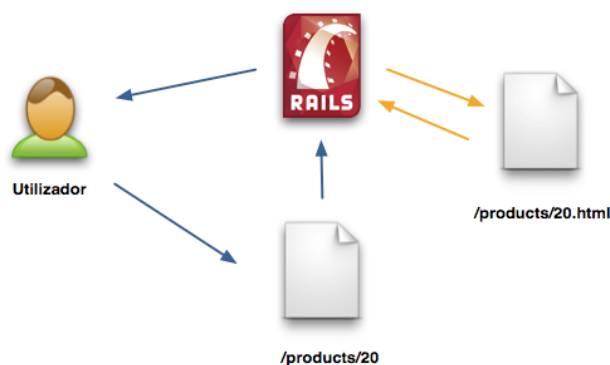


Figura 6 – Mecanismo de *action caching*, servindo um ficheiro HTML guardado em disco (*cache hit*).

O mecanismo de *fragment caching* é usado em casos onde não é possível guardar-se uma cópia completa da vista. Por exemplo se houver blocos de informação que não dependem apenas de um controlador. Assim, pode-se guardar apenas fragmentos dessa *view*. Uma *view* pode ser composta por diversos fragmentos, que são carregados, se necessário, como mostrado na Figura 7. Esses fragmentos podem, ao mesmo tempo, ser usados em diferentes páginas da aplicação.



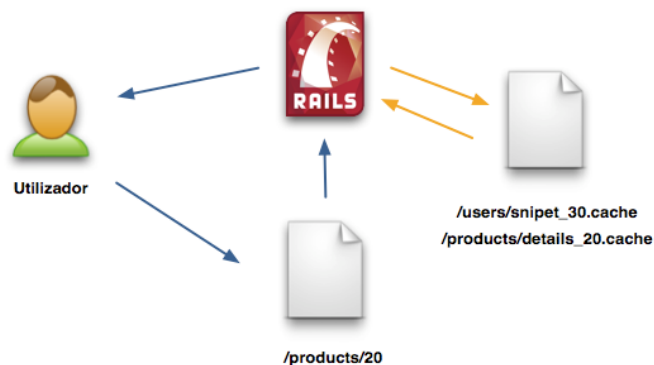


Figura 7 – Mecanismo de fragment caching, servindo diversos fragmentos guardados em disco (cache hits).

Criar versões em *cache* das páginas é apenas metade da equação. Se o conteúdo inicialmente utilizado para gerar estas páginas for alterado a versão em *cache* ficarão desactualizadas, havendo necessidade de um mecanismo para tratar a sua expiração. Assim, podem usar-se duas alternativas para a expiração dos conteúdos em *cache*: explicitamente ou implicitamente.

### 10.1. Expiração Explícita

O modo para remover páginas em *cache* de mais baixo nível é utilizando chamadas ao método `expire_page` e `expire_action` sempre que necessário. Por exemplo, no nosso controlador podemos ter uma acção que permite criar um artigo e outra que permite actualizar o conteúdo desse artigo. Quando se cria o artigo a página com a lista de artigos fica obsoleta, havendo necessidade de uma chamada `expire_page` para essa página. Ao actualizar-se um artigo, imaginando que a listagem de artigos permanece inalterada, há sempre a necessidade de remover a versão em *cache* desse artigo em particular. Assim, na actualização de um artigo é necessário incluir uma chamada a `expire_action`.

### 10.2. Expiração Implícita

Os métodos `expire_` acompanhando todas as acções que necessitem de *cache* é simples e eficaz para pequenas aplicações, mas torna-se mais complicado à medida que a aplicação cresce. Uma alteração num controlador pode afectar as páginas em *cache* de outro. A lógica contida em *helpers*, que não devia estar dependente das páginas HTML, necessita de interagir e ter em conta a expiração de páginas.

Felizmente, a *framework* Rails tem um mecanismo que permite simplificar o processo - *sweepers*. Um *sweeper* é um observador de um modelo, que, quando algo de significativo acontece, expira a *cache* associada aos dados desse modelo.

## 11. Deployment

A passagem de uma aplicação para produção (*deployment*) é uma tarefa que tende a ser mais complicada à medida que a aplicação cresce, principalmente se já tiver uma versão anterior a funcionar. Isto torna-se ainda mais complexo se a aplicação for distribuída por vários servidores.

O **Capistrano** é um utilitário que se integra com Rails. Dando uma 'receita' simples ao programa descrevendo a topologia da aplicação em produção conseguimos poupar todas as dores de cabeça envolvidas no processo tendo apenas que escrever um comando. Além disso permite ainda controlo de versões, sendo igualmente fácil voltar a ter em produção uma versão anterior da nossa aplicação.

# Anexo C

---

## Desenvolvimento ágil

### 1. Filosofias

A utilização de Ruby on Rails está frequentemente associada à utilização de um desenvolvimento ágil de *software*. O desenvolvimento ágil, tal como qualquer metodologia de desenvolvimento de *software*, providencia uma estrutura conceptual para reger projectos de engenharia de software e segue os seguintes princípios:

- garantir a satisfação do cliente entregando rápida e continuamente versões funcionais;
- versões funcionais são entregues frequentemente (em semanas) e são a principal medida de progresso do projecto;
- mesmo mudanças tardias dos requisitos do projecto são bem-vindas;
- cooperação constante entre clientes ou utilizadores e programadores;
- simplicidade;
- rápida adaptação a mudanças;

Os métodos ágeis buscam a adaptação rápida a mudanças da realidade. Quando uma necessidade de um projecto muda, uma equipa adaptativa mudará também, em contraste com uma equipa preditiva que coloca o planeamento do futuro em detalhe. Assim, o desenvolvimento ágil tem pouco em comum com o tradicional modelo em cascata, uma das metodologias com mais ênfase no planeamento, onde o progresso é geralmente medido em termos de entrega de artefactos - especificação de requisitos, documentos de projecto, planos de testes, revisões de código e outros.

O modelo em cascata resulta de uma substancial integração e esforço de teste para alcançar o fim do ciclo de vida, um período que tipicamente se estende por vários meses ou anos. Os métodos ágeis, pelo contrário, produzem um desenvolvimento completo e teste de funcionalidades num período de poucas semanas ou meses. Algumas equipas ágeis usam o modelo em cascata em pequena escala, repetindo o ciclo de cascata inteiro em cada iteração.

A maioria dos métodos ágeis partilha a ênfase no desenvolvimento iterativo e incremental para a construção de versões implantadas do software em curtos períodos de tempo mas os períodos de tempo são medidos em semanas e não em meses, e a realização é efectuada de uma maneira altamente colaborativa.

## 2. Extreme Programming

Uma metodologia ágil bastante comum é a *Extreme Programming* (XP). XP é uma metodologia ideal para equipas de tamanho pequeno que desenvolvem *software* sujeito a requisitos vagos ou que mudam rapidamente. Quanto mais incerto for o rumo do projecto maior o valor estratégico desta aproximação, que permite lidar facilmente com o problema com custo mínimo.

Tipicamente, temos um custo associado a mudanças nos requisitos como os apresentados na Figura 8. O gráfico aproxima-se de uma exponencial, quanto mais tarde forem feitas essas mudanças. Um problema que pode custar no total um Euro a resolver quando encontrado na fase de análise pode custar centenas se o projecto estiver já na sua fase final.

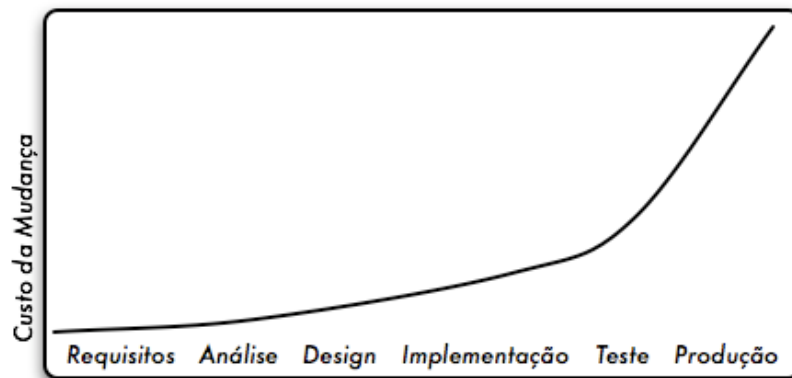
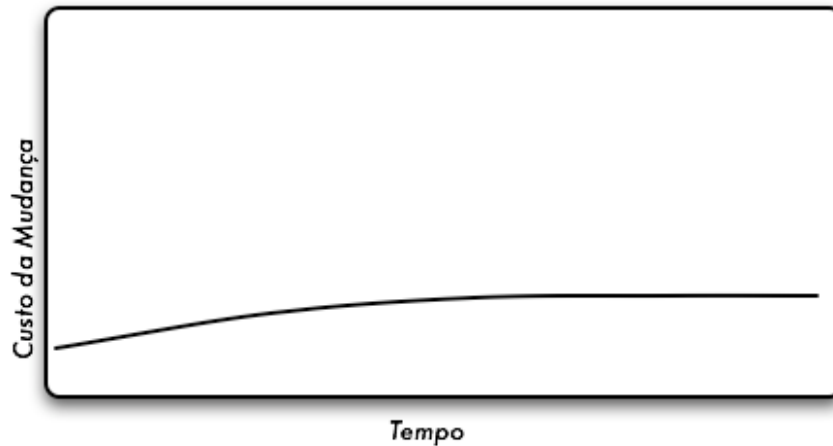


Figura 8 – Custo da mudança sem utilizar Extreme Programming.

O objectivo de XP é aproximar este gráfico, tanto quanto possível do da Figura 9. Se o custo associado a mudanças crescer lentamente com o tempo a acção a tomar perante elas também será completamente diferente de quando o custo cresce exponencialmente.



*Figura 9 – Custo da mudança ideal.*

Grandes decisões podem ser tomadas em qualquer altura do projecto, alcançando-se assim um resultado final mais satisfatório.

*Extreme Programming* rege-se por 5 princípios básicos:

- **comunicação** – na construção de software é necessário haver colaboração entre os utilizadores e/ou clientes com os programadores, muitas vezes por comunicação verbal.
- **simplicidade** – começa-se com a solução mais simples para o problema adicionando-se, posteriormente, novas funcionalidades. Foca-se assim no desenvolvimento para hoje e não para amanhã, daqui a uma semana ou um mês. A simplicidade também facilita a comunicação.
- **feedback** – tem de haver não só um feedback periódico da parte do cliente mas também da equipa de desenvolvimento, respondendo com estimativas de tempo quando o cliente surge com um novo requisito.
- **perseverança** – permite que o programador se sinta confortável com a reestruturação do que fez até ao momento tendo que apagar algum código obsoleto independentemente do trabalho que deu a escreve-lo.
- **respeito** – os membros da equipa devem respeitar-se mutuamente não devendo proceder a alterações que interfiram de algum modo com o trabalho dos colegas.

Na metodologia XP o desenvolvimento é conduzido por testes. Primeiro fazem-se os testes e só depois se escreve código. Enquanto algum teste falhar então os requisitos

para os quais foram escritos não estão cumpridos. Por esta razão está intimamente associado a *Test Driven Development* como descrito no próximo capítulo.

### 3. Test Driven Development (TDD)

Normalmente, quanto mais rapidamente é necessário desenvolver código menos testes são feitos para acelerar o processo de desenvolvimento. No entanto, a escassez de testes tende a gerar menos produtividade e a tornar o código menos robusto. Menor produtividade implica mais pressão, criando-se assim um ciclo vicioso.

Segundo alguns autores é necessário testar todas as potenciais fontes de problema. Ou seja, não é necessário testar nada que não possa gerar problemas. Isto implica alguma subjectividade quando se avaliam as potenciais fontes, sendo obrigatório haver um entendimento muito grande se tratar de uma equipa e não de um único indivíduo.

*Test Driven Development* (TDD) é uma característica da *Extreme Programming* e permite atribuir à mesma pessoa funções de desenvolvimento e teste em simultâneo. Seguindo uma metodologia TDD é necessário escrever um conjunto exaustivo de testes, associados a pequenos detalhes funcionais da aplicação em questão.

Com TDD este risco é eliminado, uma vez que antes da implementação de código vem a escrita dos testes para todas as novas funcionalidades a ser implementadas. Estes testes vão falhar inicialmente, havendo depois uma escrita de código com vista a que a aplicação passe nos testes.

Após a escrita dos testes para determinadas funcionalidades e de desenvolvido o código para passar nesses testes volta-se ao início. Assim, a metodologia TDD segue um ciclo como o apresentado na Figura 10.

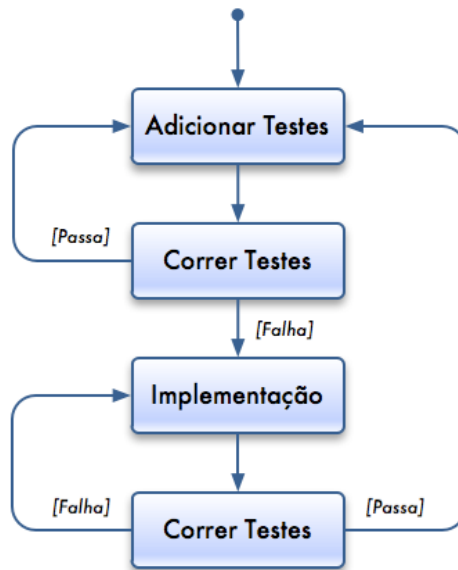


Figura 10 – Ciclo de escrita de testes seguindo Test Driven Development.

Todo o trabalho desenvolvido é assim mantido numa só equipa, com controlo absoluto de todo o código desenvolvido em paralelo aos testes da aplicação em pequenos ciclos.

## 4. Behaviour Driven Development (BDD)

*Behaviour Driven Development* (BDD) é geralmente apresentado como uma evolução de TDD em que se deve afastar o mais possível do conceito tradicional de testes.

Geralmente o acto de testar *software* é visto de forma negativa e leva à desmotivação por parte de quem tem de fazer testes, por falta de tempo ou por desvalorização da sua importância. Por esta razão os testes devem ser vistos como um planeamento daquilo que se quer implementar, do ponto de vista do comportamento da aplicação. No fundo podem traduzir-se como especificação e não como testes propriamente ditos.

A principal diferença para TDD é o vocabulário. Em vez de classes de teste tem-se contextos e vem vez de se escrever métodos começados com “test” começa-se com “should”.

## 5. TDD e BDD em Rails

Um dos aspectos mais proeminentes da *framework* Ruby on Rails é a utilização da própria linguagem Ruby. Sendo uma linguagem dinâmica tem uma grande flexibilidade, conveniência e poder, mas com alguns custos. Como não há compilação não é possível detectar alguns erros, incluindo erros relativamente comuns e gralhas. A presença de um mecanismo de testes torna-se assim extremamente importante.

A *framework* Ruby on Rails está bem preparada para a realização de testes, sendo criada uma directoria “test” para cada projecto, onde são gerados ficheiros de teste, automaticamente, sempre que é criado um novo modelo. Os testes são divididos em três tipos:

- **Testes unitários:** sobre os modelos da aplicação. Servem para verificar que os modelos fazem o que é suposto e que todas as associações estão correctas.
- **Testes funcionais:** aplicados aos controladores e à interacção entre modelos. São fundamentalmente exercícios à interface usando pedidos HTTP entre os diferentes controladores.
- **Testes de integração:** mais gerais, guiados por ‘histórias’, que verificam as interacções entre várias acções suportadas pela aplicação, envolvendo diferentes controladores.

Os testes são escritos em Ruby, havendo um conjunto de funções simples já definidas na própria linguagem, tendo sido estendidas com funções específicas de Rails (para testes directamente ligados à *framework* como rotas, redireccionamentos, etc.).

Utilizando a biblioteca de testes instalada por omissão pode escrever-se um teste como por exemplo:

```
def test_length
  string = 'abc'
  assert_equal(3, string.length)
end
```

que, para quem está habituado a escrever coisas como `5.days.from_now`, não é tão legível nem intuitivo. Para utilizar os testes em Ruby tem que se aprender a lidar com uma sintaxe completamente nova, bem como uma legibilidade de código inferior.



Por esta razão, muitas vezes opta-se por migrar os testes para bibliotecas de testes que permite expressar os mesmos testes de forma mais legível. RSpec é uma dessas bibliotecas, onde o teste indicado anteriormente pode ser escrito como:

```
context "A String" do
  specify "Should return the number of characters it contains" do
    string = "abc"
    string.length.should_equal 3
  end
end
```

que, apesar de ter bastante mais texto é quase que uma prosa em Inglês. Este novo tipo de aproximação aos testes corresponde ao BDD em Rails.

Existe já um leque bastante grande de bibliotecas de testes além da instalada por omissão, seguindo uma filosofia próxima de RSpec. No entanto, tratam-se de implementações em tudo semelhantes, com pequenas diferenças em aspectos específicos. Tomando como exemplo uma dessas bibliotecas o teste anterior vem então:

```
context "A String" do
  should "return the number of characters it contains" do
    string = "abc"
    assert_equal string.length, 3
  end
end
```

# Anexo D

## Especificação da aplicação de estudo

Nos capítulos seguintes é apresentada a especificação da aplicação de teste utilizada no estudo comparativo de diversas *frameworks*. Esta aplicação é muito simples, servindo para testar a facilidade de utilização e rapidez de desenvolvimento nessas *frameworks*.

### 1. Requisitos

Na tabela seguinte apresentam-se os requisitos da aplicação utilizada no estudo.

| Identificação | Nome  | Descrição   |
|---------------|---|---|
| R01           | Idioma único  | Toda a interface é em inglês.   |
| R02           | Base de dados   | Todos os registos são guardados numa base de dados em MySQL 5.  |
| R03           | Gestão de utilizadores                                  | O utilizador pode adicionar, editar e apagar utilizadores ( <i>users</i> ). Deve poder ainda listar todos os utilizadores e visualizar a informação de um utilizador em particular. |
| R04           | Gestão de comentários                                   | O utilizador pode adicionar, editar e listar comentários ( <i>comments</i> ).   |
| R05           | Associação de comentários                               | Os comentários estão sempre associados a um utilizador, não sendo possível a criação de um comentário sem utilizador.   |
| R06           | Os comentários de um utilizador são apagados em cascata | Quando um utilizador é apagado todos os comentários a ele associados são apagados.  |
| R07           | Apresentação do número de comentários                   | O número de comentários de cada utilizador deve ser apresentado na lista.   |
| R08           | Apresentação dos comentários                            | Deve ser apresentada a lista de comentários nos detalhes de utilizador.   |
| R09           | Campos obrigatórios                                     | Todos os campos são obrigatórios devendo ser validados aquando a criação de novos utilizadores ou comentários.  |

Tabela 1 – Tabela de requisitos da aplicação de estudo.

## 2. Diagramas de casos de uso

A primeira componente desta aplicação é a gestão de utilizadores. Os utilizadores podem ser listados, adicionados, removidos e editados. As três últimas operações interagem com a base de dados.

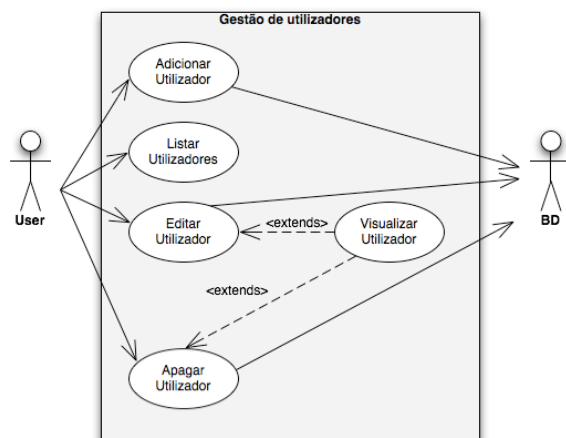


Figura 11 – Diagrama de Casos de Uso para a gestão de utilizadores.

A segunda componente é a de gestão de comentários. Os comentários podem ser listados, adicionados, e editados. Os comentários devem ser associados ainda a um utilizador.

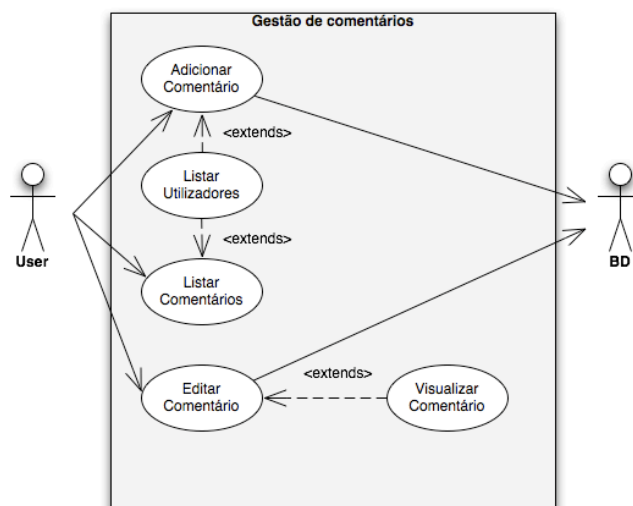


Figura 12 – Diagrama de Casos de Uso para a gestão de comentários.

### 3. Modelo de dados

O modelo de dados da aplicação é composto apenas por duas tabelas. Todos os campos são obrigatórios.

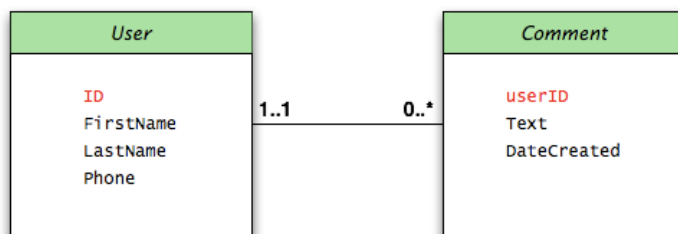


Figura 13 - Modelo de dados para a aplicação de teste das diferentes frameworks.

### 4. Protótipo

Nas páginas seguintes estão listados os ecrãs exemplificando o modo como é apresentada a informação e que funcionalidades existe em cada um.



Figura 14 – Listagem de utilizadores com contagem de comentários associados a cada um.

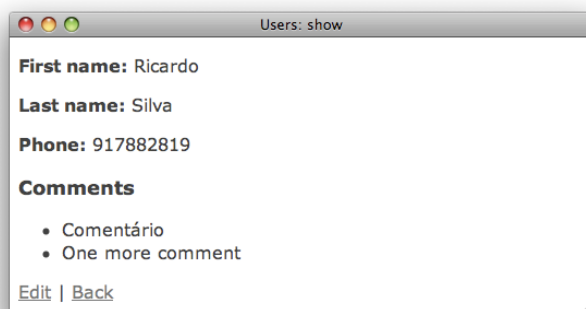


Figura 15 – Detalhes de um utilizador. Os comentários aparecem listados.

Users: edit

## Editing user

First name  
Ricardo

Last name  
Silva

Phone  
917882819

[Show](#) | [Back](#)

Figura 16 – Edição de um utilizador já existente.

Users: new

## New user

First name

Last name

Phone

[Back](#)

Figura 17 – Formulário de criação de um novo utilizador.

Comments: index

## Listing comments

| User | Text             |   |
|------|------------------|---|
| 1    | Comentário       | <a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a> |
| 1    | One more comment | <a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a> |

[New comment](#)

Figura 18 – Listagem dos comentários existentes.

The image shows a browser window titled "Comments: new". The main heading is "New comment". Below the heading, there is a "User" label followed by a text input field. Underneath that is a "Text" label followed by a large text area for writing the comment. At the bottom left of the form, there is a "Create" button and a "Back" link.

*Figura 19 – Adição de um novo comentário.*

# Anexo E

## Especificação da aplicação RoomBlick

A aplicação desenvolvida envolveu a escrita de uma especificação inicial, que foi depois complementada na segunda fase do estágio, com levantamento de novos requisitos e está apresentada neste anexo.

### 1. Requisitos

A tabela seguinte apresenta os requisitos da aplicação.

| Identificação | Nome                           | Descrição   |
|---------------|--------------------------------|---|
| R01           | Internacionalização            | A aplicação deve ter suporte para uma internacionalização de modo simples.  |
| R02           | Escolha da língua              | O utilizador deve ter a possibilidade de mudar a língua de modo simples e imediato.   |
| R03           | Inserção de datas              | Todas as datas devem ter ajudas visuais aquando a sua inserção.   |
| R04           | Perfis de utilizador           | A aplicação deve ter diferentes perfis de utilizador detalhados na tabela seguinte.   |
| R05           | Área privada                   | Os utilizadores registados devem ter acesso a uma área privada.   |
| R06           | Autenticação                   | Para visualização de conteúdos privados é necessária uma autenticação prévia. A autenticação é feita com <i>email</i> e <i>password</i> . Como alternativa pode ser usado OpenID (ver R07). |
| R07           | Autenticação alternativa       | Para visualização de conteúdos privados poderá fornecer-se um endereço de OpenID válido.  |
| R08           | Recuperação de <i>password</i> | Os utilizadores registados devem poder recuperar a <i>password</i> de autenticação no sistema com recurso a uma <i>hash</i> enviada para o <i>email</i> .                                   |
| R09           | Editar perfil                  | Todos os utilizadores registados podem editar o seu perfil na sua área privada. Aí podem, além de editar qualquer informação submetida no registo, associar                                 |

|     |                                     |  |
|-----|-------------------------------------|--|
|     |                                     | uma imagem.  |
| R10 | Adição de um anúncio                | Um anúncio só pode ser adicionado por utilizadores registados acedendo à sua zona privada.   |
| R11 | Visibilidade de um anúncio          | Um anúncio só será visível em zona pública quando correctamente validado.  |
| R12 | Validação de um anúncio             | Para validar um anúncio um utilizador tem de enviar um SMS com recurso a uma <i>hash</i> gerada automaticamente.   |
| R13 | Expiração de um anúncio             | O sistema tem de permitir definir um tempo de validade para um anúncio, depois do qual terá de haver nova activação. Tem também de ser possível dar uma margem temporal para que os anúncios permaneçam inactivos.                         |
| R14 | Detalhes de anúncio                 | Qualquer utilizador poderá consultar os detalhes de um anúncio mesmo que não registado.  |
| R15 | Comentários de anúncio              | Todos os anúncios devem permitir a adição de comentários por utilizadores registados. Os comentários devem aparecer listados na página de detalhes.  |
| R16 | Imagens associadas a um anúncio     | O utilizador pode associar aos seus anúncios um máximo de 5 imagens. Estas imagens têm no máximo 2Mb e são guardadas duas cópias. Uma com dimensões máximas de 110x88px e outra 220x176px.   |
| R17 | <i>Slideshow</i> de imagens         | Quando na página de detalhes do anúncio deve ser visível um <i>slideshow</i> dinâmico com as imagens a ele associadas.   |
| R18 | Edição de um anúncio                | Um utilizador pode, a qualquer momento, editar os detalhes de um anúncio na sua área privada.  |
| R19 | Imagem identificativa de um anúncio | Todos os anúncios têm uma imagem que o identifica e que aparece nas listagens de anúncios.   |
| R20 | Imagens identificativas por omissão | Todos os anúncios e utilizadores têm por omissão uma mesma imagem que os identifica no caso de não ser associada outra.  |
| R21 | Reordenação de imagens              | Quando é editado um anúncio deve ser possível definir a ordem das imagens a ele associadas. Esta ordem tem influência na sequência de imagens do <i>slideshow</i> e na imagem identificativa do anúncio, que deve ser a primeira da lista. |
| R22 | Formato de imagens                  | O sistema deve aceitar imagens nos   |



|     |                                      |  |
|-----|--------------------------------------|--|
|     |                                      | seguintes formatos: PNG, GIF e JPEG.   |
| R23 | Paginações                           | A listagem de anúncios e de comentários a um anúncio devem ser paginados de modo a não se criar um <i>scroll</i> demasiadamente grande no <i>browser</i> .   |
| R24 | Georreferenciação dos quartos        | Todos os quartos têm de ser guardados com uma latitude e longitude. A localização do quarto deve aparecer na página de detalhes de um anúncio com ajuda visual de um mapa.   |
| R25 | Inserção de coordenadas de um quarto | A inserção da localização de um quarto deve ser fácil e amigável permitindo a tentativa de mapear automaticamente uma morada e um mecanismo de <i>drag and drop</i> de uma marca num mapa real. Esse mapa deve poder ser ampliado e arrastado. |
| R26 | Edição das coordenadas de um quarto  | A edição da localização de um quarto deve ser tão fácil quando a sua primeira inserção.  |
| R27 | Pesquisa avançada                    | A aplicação deve permitir pesquisar anúncios de quartos com base nos diferentes parâmetros de um anúncio.  |
| R28 | Pontos de interesse                  | Deve ser possível pesquisar quartos pela proximidade a pontos de interesse.  |
| R29 | Inserção de pontos de interesse      | Os pontos de interesse devem poder ser inseridos individualmente (de forma semelhante aos quartos) ou em listas de pontos de interesse. Esta inserção só pode ser feita por administradores.   |
| R30 | Categorias dos pontos de interesse   | Os pontos de interesse devem ser agrupados em categorias. (ex: Universidade, Escola, Estação de Coimbra, Mercado, etc.)  |
| R31 | Mensagens privadas                   | Todos os utilizadores registados devem ter acesso a uma caixa de mensagens privadas onde podem enviar e receber mensagens de outros utilizadores ou receber avisos do sistema.   |
| R32 | Notificações por email               | Os utilizadores devem receber notificações por email quando recebem mensagens de sistema na caixa de entrada das mensagens privadas da aplicação.  |
| R33 | Pesquisa rápida                      | Na página de entrada deve haver uma versão minimalista do motor de pesquisa.   |

Tabela 2 – Tabela de requisitos da aplicação RoomBlick.

Na Tabela 3 está uma tabela com os dois tipos de perfil que a aplicação suporta.

| Nome  | Permissões   |
|-------|--|
| User  | Pode editar o perfil, adicionar/listar/editar/validar anúncios, adicionar comentários, receber e enviar mensagens privadas. Pode ainda recuperar a password. |
| Admin | Tem todas as permissões dos utilizadores com perfil 'User' e pode ainda gerir utilizadores, pontos de interesse e apagar/editar qualquer anúncio.            |

Tabela 3 – Perfis de utilizador

## 2. Diagramas de casos de uso

Nos diagramas seguintes apresentam-se requisitos funcionais do sistema de anúncios. Todos os utilizadores podem criar anúncios, sendo que para clarificação dos diagramas, assumem um papel de “anunciantes” quando são tarefas relativas apenas aos seus anúncios.

### Registo e autenticação

Todos os utilizadores registados têm acesso a conteúdo privado. O utilizador pode ainda alterar todos os dados constantes no seu perfil quando bem entender. A informação dos utilizadores é guardada em base de dados.

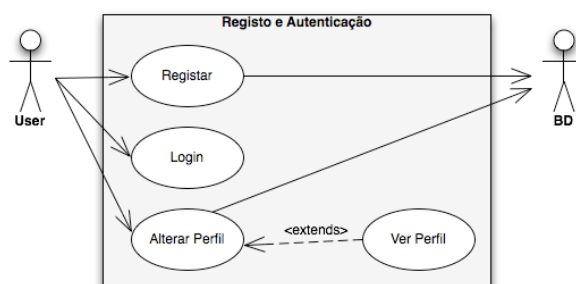


Figura 20 – Diagrama de Casos de Uso para o registo e autenticação de um utilizador.

### Recuperação de password

Para recuperação de password é enviado um email com uma *hash* gerada automaticamente, que leva a um endereço usando essa *hash*, onde o utilizador pode redefinir a sua palavra-chave. Esta funcionalidade está associada a um *link* com o nome “Esqueceu a sua password?”.

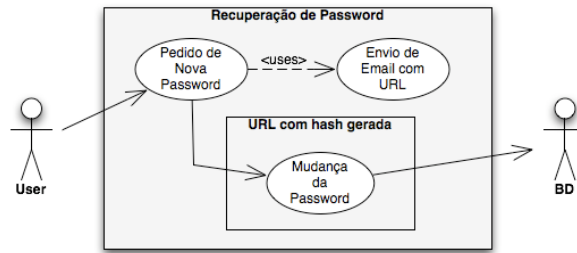


Figura 21 – Diagrama de Casos de Uso para a recuperação de password.

## Gestão de um anúncio

Para validação do anúncio tem de ser enviada uma SMS de forma a que este passe a ser visível na listagem pública de anúncios. Um anúncio deverá ter uma validade que quando finda exige envio de nova SMS.

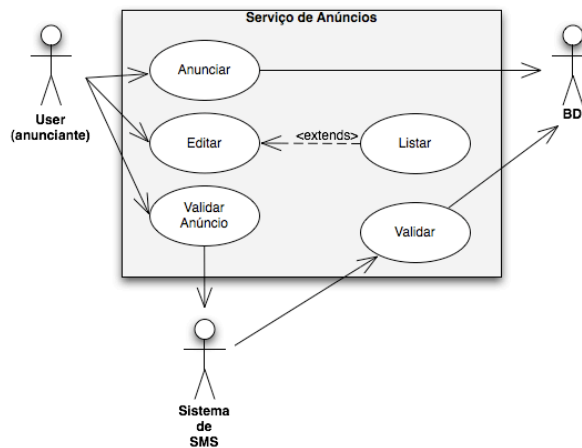


Figura 22 – Diagrama de Casos de Uso para a gestão de um anúncio.

## Arrendamento e comentários

Quando um utilizador tenciona fazer um pedido de arrendamento o anunciante desse anúncio deve receber uma notificação acerca da intenção do primeiro. Quando efectua uma pesquisa o utilizador deve ser capaz de filtrar os resultados por diversos parâmetros. Esta filtragem deve ser opcional.

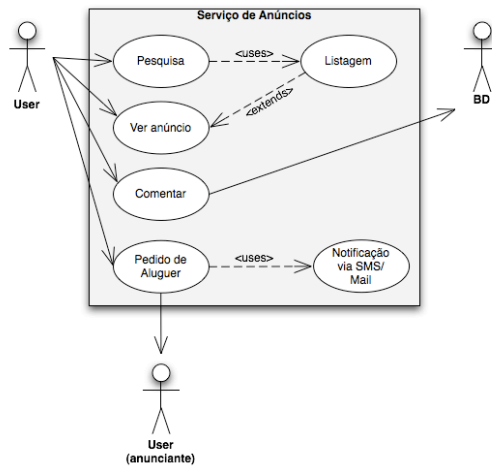


Figura 23 – Diagrama de Casos de Uso para arrendamento e adição de comentários.

### 3. Protótipo

Na Figura 24 está o aspecto da página de entrada nesta aplicação quando o utilizador não efectuou a sua autenticação. Na área central pode ser feita uma pesquisa rápida e login na aplicação. Na Figura 25 mostra-se o mesmo ecrã depois de efectuado o login.



Figura 24 – Página inicial sem autenticação.



Figura 25 – Página inicial com autenticação.

O formulário de registo apresentado na Figura 26 apresenta os campos obrigatórios com um sinal vermelho para que o utilizador consiga aperceber-se de quais são esses campos com maior facilidade. O campo de data de nascimento faz activar um *popup* com um calendário para uma escolha mais amigável da data.

Figura 26 – Formulário de registo.

O perfil dos utilizadores pode depois ser editado num ecrã como o demonstrado na figura seguinte. Aí pode ainda ser feita uma associação a uma imagem enviada pelo utilizador.

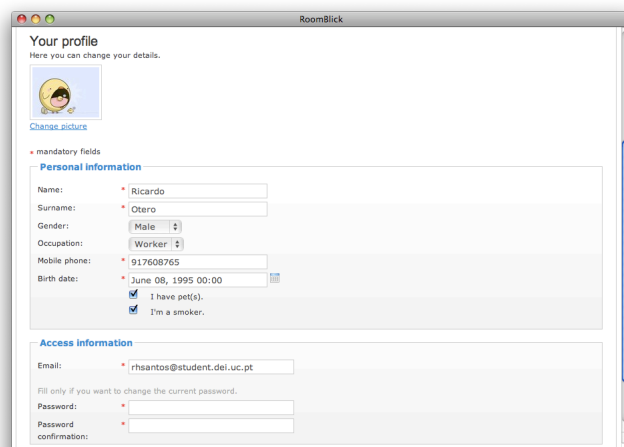


Figura 27 – Formulário de edição de perfil.

Na Figura 28 está apresentada a área disponível apenas para administradores onde se pode fazer a gestão dos pontos de interesse.

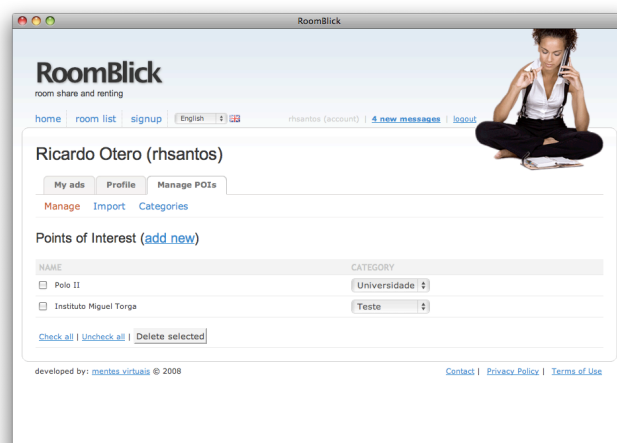


Figura 28 – Página de gestão de pontos de interesse.

Na Figura 29 é apresentado o formulário de importação de uma lista de pontos de interesse também disponível a administradores.

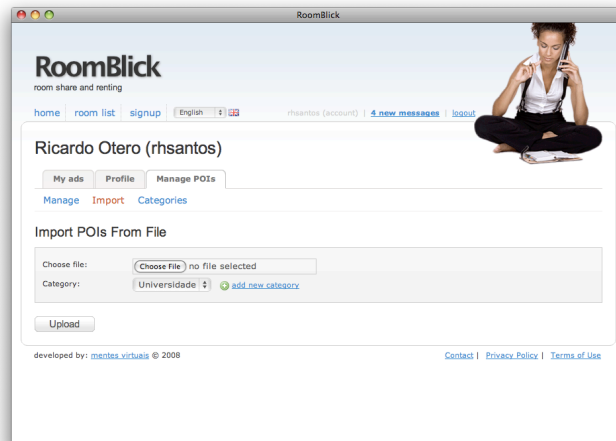


Figura 29 – Página de importação de uma lista de pontos de interesse.

Na Figura 30 está a página de administração das categorias dos pontos de interesse. Esta área também é reservada a utilizadores com privilégio de administrador.

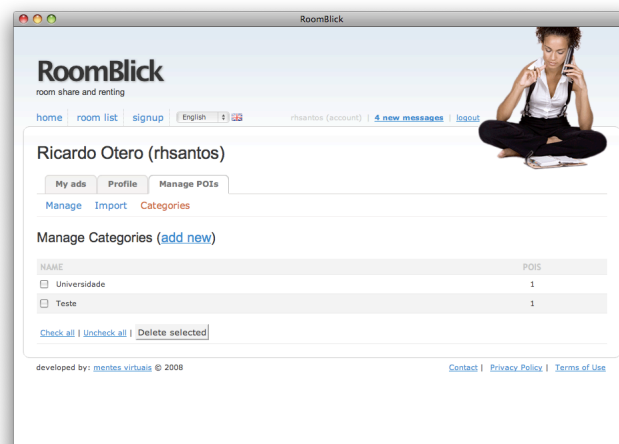


Figura 30 – Página de gestão de categorias de pontos de interesses.

Nas duas figuras seguintes apresenta-se os ecrãs de autenticação com *username* e *password* e a alternativa, feita com OpenID.

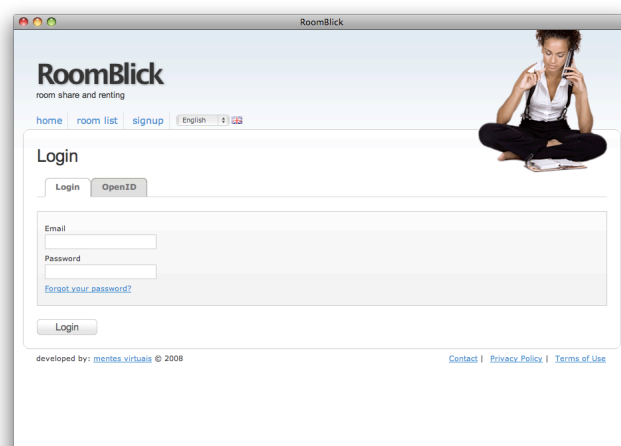


Figura 31 – Formulário de login com email e password.

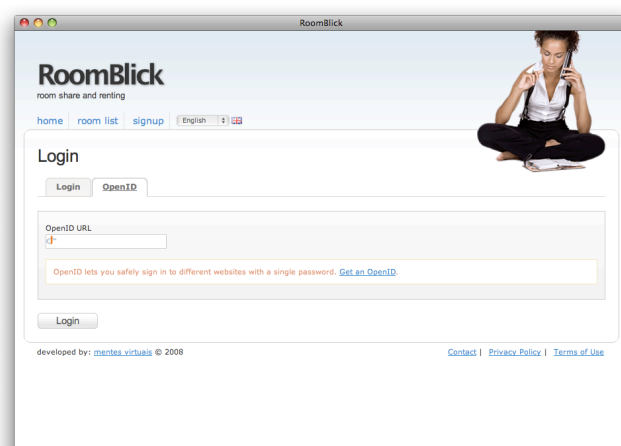


Figura 32 – Formulário de login com OpenID.

Na Figura 33 é apresentada a área de mensagens privadas disponível para todos os utilizadores registados. Esta área funciona como uma caixa de mensagens privada na aplicação de onde se podem receber e enviar mensagens para outros utilizadores e receber notificações do sistema. Nas duas figuras seguintes está apresentada a funcionalidade de pesquisa de quartos.



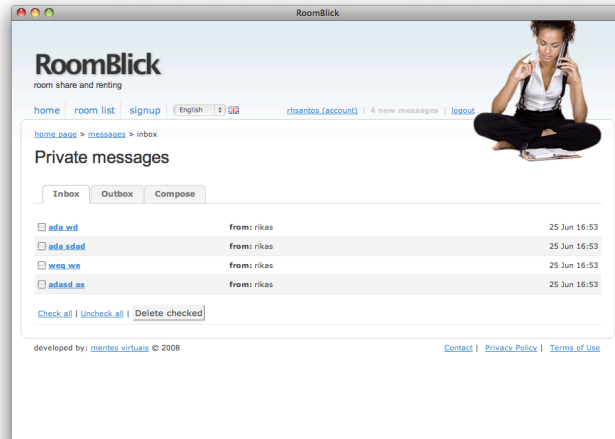


Figura 33 – Listagem de mensagens pessoais.



Figura 34 – Listagem de anúncios.

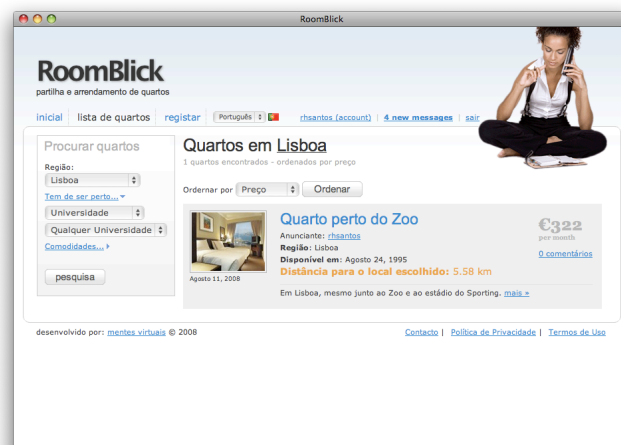


Figura 35 – Resultado de uma pesquisa de anúncios.

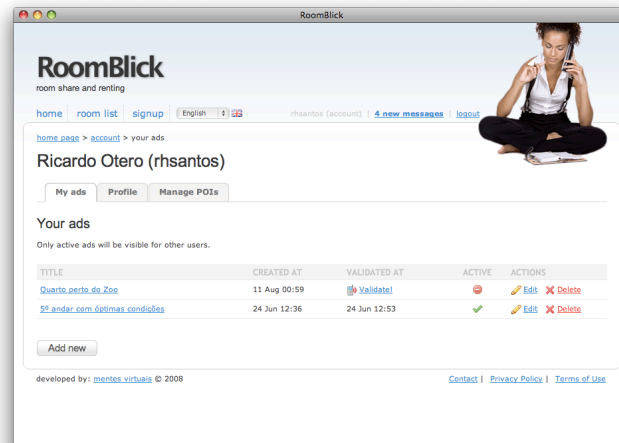


Figura 36 – Listagem de anúncios de um utilizador na sua área privada.

Na Figura 37 pode ver-se o ecrã de edição de um anúncio, com principal ênfase na disponibilidade das imagens a ele associadas, que podem ser reordenadas e apagadas.

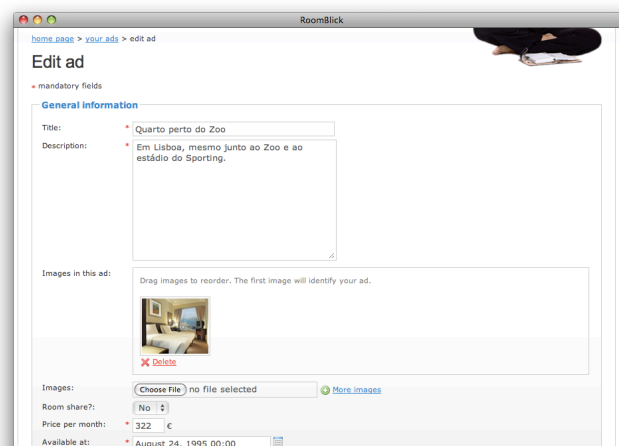


Figura 37 – Edição de um anúncio.

Na Figura 38 apresenta-se o ecrã de edição da localização de um quarto com ajuda de um mapa em Javascript.

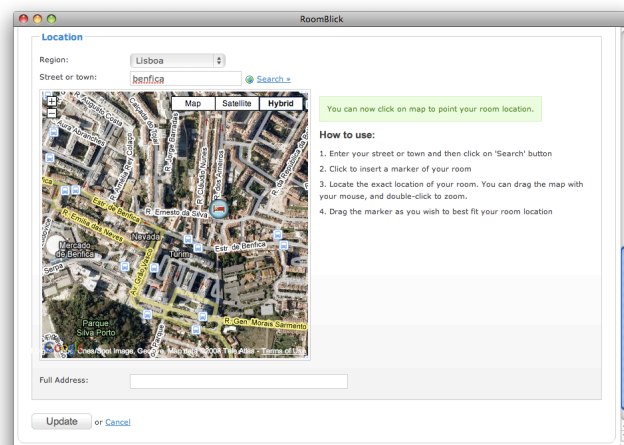


Figura 38 – Edição da localização de um quarto.

# Anexo F

---

## Estado da arte em aplicações de arrendamento

### 1. Introdução

O mercado de arrendamento ou partilha de quartos *on-line* não é um mercado inexplorado. Assim, existem algumas soluções que foram alvo de estudo preliminar no sentido de apurar funcionalidades que poderiam alargar o leque de requisitos do *RoomBlick*. As soluções existentes deveriam ser direccionadas para a população portuguesa, dado que, numa primeira fase, a aplicação será condicionada a Portugal.

### 2. Bquarto

(<http://www.bquarto.pt>)

Esta é uma aplicação desenvolvida por uma empresa nacional, que contempla quartos apenas em Portugal. Permite pesquisar pessoas que querem um quarto para arrendar ou que disponibilizem arrendamento. A aplicação disponibiliza uma pesquisa de quartos sem ser necessário um registo no sistema mas essa pesquisa apenas permite filtrar os quartos por zona do país, não havendo qualquer outra opção. Existem duas áreas distintas – uma para quem tem quartos para arrendar e outra para quem procura quartos.

#### 2.1. Interface

A interface é confusa, sem coerência de umas secções para outras, com a informação desorganizada e muitas vezes, mesmo para um utilizador experiente, é difícil compreender como efectuar determinadas operações. A aplicação apresenta, logo na página de entrada, 48 erros de HTML, ignorando as especificações técnicas em vigor. Além disso utiliza tabelas para o *layout* das páginas, uma prática altamente desaconselhada pelos mais diversos motivos.

Na Figura 39 está apresentada a página de entrada desta aplicação. Sem que o utilizador se aperceba, a opção da região de Portugal na caixa azul vai limitar a visualização dos anúncios em todas a aplicação, tendo que se escolher outra localização na página de entrada para conseguir ver-se quartos em regiões diferentes da inicialmente seleccionada.



Figura 39 – Aspecto da página de entrada de BQuarto.pt.

Apesar de se ter que fornecer muita informação pessoal quando se pretende visualizar ou criar anúncios, depois essa informação não é apresentada de forma clara, sendo muitas vezes omitida ou imbricada de forma pouco perceptível, servindo a

Figura 40 como exemplo.

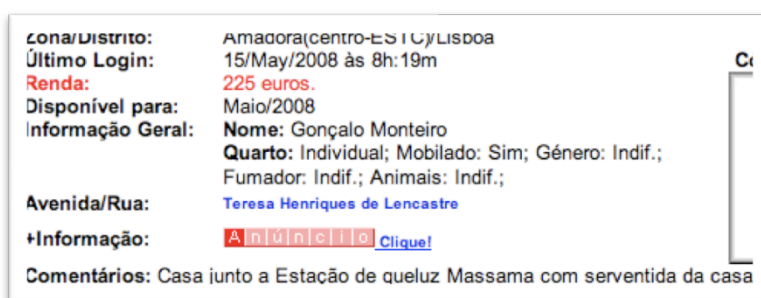


Figura 40 – Detalhes de um anúncio mostrando falta de organização na informação.

Um problema recorrente é a presença de *links* sem significado explícito que muitas vezes levam a um resultado inesperado. Na

Figura 40 a imagem a vermelho, cujo significado não é claro, é uma ligação que leva a uma página de colocação de novo anúncio, não tendo sequer relação com a etiqueta “+Informação” apresentada. Esta aplicação só permite adicionar uma fotografia pequena (130x130) por cada quarto, o que não é suficiente para apresentar o aspecto dos quartos. Por último importa referir que as rendas aparecem apenas nos detalhes de cada quarto obrigando os utilizadores a ver os detalhes de todos os quartos para fazer uma comparação rápida por preço.

## 2.2. Utilizadores

Na aplicação não é possível ver quão antigos são os anúncios, havendo apenas uma “data de último login” do responsável. No entanto todos os anúncios são recentes (criados, no máximo, há 1 mês ). Assim, o número de utilizadores activos, neste espaço temporal, pode ser estimado somando os anúncios de todas as regiões e temos então:

- “Tenho Quartos” - 207 em todo o país;
- “Procuo Quartos” - 3382 em todo o país.

Na página inicial pode-se também ver quantos utilizadores estão *on-line* e varia normalmente entre 10~15 durante o dia.

## 2.3. Campos de registo

Nesta aplicação o registo de um novo anúncio implica o fornecimento dos seguintes dados:

- Renda mensal
- O valor da renda mensal inclui (sim/não)

|      |               |          |
|------|---------------|----------|
| Água | Electricidade | Internet |
| Gás  | Telefone      | TV cabo  |

- Disponível a partir de...
- Duração mínima de arrendamento de...
- Sobre o quarto (sim / não)

|            |                 |
|------------|-----------------|
| Individual | Chão de madeira |
| Mobilado   | Tomada telefone |

|              |            |
|--------------|------------|
| Tomada de TV | WC privado |
|--------------|------------|

- Tamanho (m2)
- Sobre a habitação e equipamentos/ espaços que pode utilizar

|                     |                  |            |             |
|---------------------|------------------|------------|-------------|
| Nº de WC            | Fogão            | Computador | Cozinha     |
| Aquecimento central | Máquina de loiça | Elevador   | Micro-ondas |
| Ar condicionado     | TV               | Piscina    | Frigorífico |
| Máquina de roupa    | TV Cabo          | Internet   | Ginásio     |

- Tipo de habitação (apartamento/vivenda/residência)
- Número total de quartos
- Quem procura para este espaço
  1. Género
  2. Ocupação
  3. Orientação
  4. Signo
  5. Idade máx. / Idade min.
  6. Pode ter animais de estimação?
  7. Pode ser fumador?

Como facilmente se pode perceber os formulários de registo são muito extensos e apresentados de forma pouco organizada não atraindo possíveis compradores.

### 3. Easyquarto

[www.easyquarto.com.pt](http://www.easyquarto.com.pt)

Ao contrário da aplicação anterior, esta é internacional, tendo um domínio dedicado a quartos em Portugal. Tem suporte para várias línguas, mas a portuguesa tem muitos erros como se pode ver na Figura 41, entre esses erros destacam-se:

- traduções mal feitas, com português do Brasil em muitas situações;
- erros ortográficos;
- frases ou títulos em inglês;
- erros gramaticais.



Figura 41 – Alguns exemplos de erros de português na aplicação “easyquarto”.

De modo análogo à aplicação anterior, também aqui se separam os anúncios em dois grupos – utilizadores que arrendam quartos e utilizadores que procuram quartos. A aplicação não permite pesquisa sem registo, tendo de se preencher um formulário extenso mesmo que o utilizador apenas pretenda ver quais os anúncios disponíveis (tendo de colocar um anúncio de “quero quarto”) o que pode levar a que muitos utilizadores nem queiram experimentar a aplicação.

### 3.1. Interface

A interface não é tão confusa como a da aplicação anterior, mas apresenta muito mais erros de HTML (583 na primeira página). Os erros na escrita em português contribuem para uma dificuldade acrescida em operações que deveriam ser simples e podem mesmo tornar-se confusas ao ponto de não ser perceptível em que área o utilizador se encontra (de procura de quartos ou de disponibilização de quartos para arrendar).

Esta aplicação utiliza a API do Google Maps para a localização dos quartos, mas depois de se testar a inserção de quatro anúncios em localizações correctas verificou-se que estes nunca eram apresentados correctamente.

Depois de efectuado o registo, já é possível pesquisar anúncios e, ao contrário da aplicação anterior, há a possibilidade de filtrar os resultados por diversos parâmetros. É também possível adicionar até 4 fotografias por quarto, de dimensões maiores que as da aplicação anterior.



### 3.2. Utilizadores

Os anúncios presentes nesta aplicação são todos de, no máximo, três meses atrás. Neste intervalo de tempo, podemos contabilizar o total de anúncios de modo a estimar qual a utilização desta aplicação, temos assim:

- “Tenho Quartos” - 420 quartos em todo o país;
- “Preciso Quartos” - 4884 quartos em todo o país.

### 3.3. Campos de registo

- Mora / não mora no apartamento
- Morada
- Renda (/mês /semana /dia)
- Disponível a partir de...
- Disponível por...
- Tipo de imóvel (apartamento / casa)
- Número de quartos
- Número de WC
- Piso
- Número de quartos disponíveis
- Tamanho da moradia
- Sobre o quarto (sim / não)

|                 |                    |                 |                 |
|-----------------|--------------------|-----------------|-----------------|
| Mobilado        | TV                 | Entrada privada | WC privado      |
| Piso de madeira | Tomada para TV     | Roupa de cama   | Ar condicionado |
| Armário         | Tomada de telefone | Carpete         |                 |

- Sobre o imóvel (sim / não)

|                   |                  |          |          |
|-------------------|------------------|----------|----------|
| Sistema de alarme | Máquina de roupa | Varanda  | Internet |
| TV por cabo       | Máquina de loiça | Porteiro | Ginásio  |
| Quintal           | Garagem          | Elevador | Piscina  |

- Sexo do novo morador
- Aceita casais?

## 4. Conclusões

Ambas as aplicações apresentam problemas evidentes a vários níveis. Estas aplicações falham na falta de cuidado com aspectos tão fundamentais como a usabilidade ou a utilização de *standards*. São de resto muito parecidas no que toca ao modelo de negócio, sendo de registo gratuito, com regalias para contas especiais (pagas).

Estas não são aplicações de utilização agradável. O facto de um utilizador ter de criar um perfil com formulários bastante extensos só para consultar os anúncios colocados desencoraja o registo de novos utilizadores. Ficam também a faltar funcionalidades essenciais como a pesquisa com diferentes parâmetros ou a possibilidade de comparar de forma mais directa os quartos apresentados, uma vez que as listagens têm pouca informação.

# Anexo G

## Integração com OpenID

Para integrar a aplicação com OpenID em primeiro lugar foi necessária a instalação da biblioteca ruby que implementa o protocolo OpenID:

```
gem install ruby-openid
```

Importa salientar que o *plugin* utilizado depende de uma versão da *gem* 2.0 ou superior. Depois de instalada a *gem* e o *plugin* "*open\_id\_authentication*", é necessário correr a *task* para criação das tabelas necessárias, para associação de OpenID's a utilizadores:

```
rake open_id_authentication:db:create
```

No ficheiro de rotas terá de ser adicionada uma nova rota, chamada *open\_id\_complete*, que vai ser utilizada para redireccionamento para o nosso controlador e acção certos, caso a autenticação no facilitador seja bem sucedida. No caso da aplicação desenvolvida essa rota aponta para */session/create*:

```
map.open_id_complete 'session', :controller => "sessions", :action =>
"create", :requirements => {:method => :get}
```

A nível visual, no formulário de *login* é necessária a inclusão de uma alternativa para utilização de OpenID:

```
<p>
  <label for="openid_url">OpenID:</label>
  <%= text_field_tag "openid_url" %>
</p>
<p>
  <%= submit_tag 'Sign in', :disable_with => "Signing in&hellip;" %>
</p>
```

O exemplo é muito simples, podendo ser refinado, mas note-se que o nome do campo de texto terá de ser o utilizado para que se possam usar funções de ajuda incluídas no *plugin*. O controlador de sessões terá de sofrer alterações para detectar

se o utilizador está a fazer *login* com OpenID ou não. Assim, o controlador passa a ter duas alternativas para o método `create`:

```
def create
  if using_open_id?
    open_id_authentication(params[:openid_url])
  else
    password_authentication(params[:email], params[:password])
  end
end
```

O método `using_open_id?` É um dos métodos dependente do nome correcto dado ao campo na *view* apresentada em cima. Se houver necessidade de *login* usando OpenID recorre-se então a um método alternativo de autenticação:

```
protected
def open_id_authentication(openid_url)
  authenticate_with_open_id(openid_url, :required => [:nickname, :email, :dob, :fullname, :gender]) do |result, identity_url, registration|
    if result.successful?
      @user = User.find_or_initialize_by_identity_url(identity_url)
      if @user.new_record?
        @user.gender = registration['gender'].equal?"M"
        @user.identity_url = identity_url
        @user.name = registration['fullname']
        (...)
        @user.save!
      end
      self.current_user = @user
      successful_login
    else
      failed_login result.message
    end
  end
end
```

Os dados pedidos podem ser: *nickname*, *email*, *dob*, *fullname*, *postcode*, *country*, *language*, *timezone* e *gender*. Poderão pedir-se todos ou apenas uma parte, consoante a necessidade da aplicação.

Quando é detectado que um utilizador ainda não efectuou *login* na nossa aplicação, os dados são guardados na base de dados e associados ao URL fornecido. Assim, o utilizador passa a possuir informação de modo semelhante aos utilizadores registados na aplicação, podendo editá-la da mesma forma. Guardar os dados de utilizador na base de dados é, no entanto, um passo opcional.

# Anexo H

## Implementação de georreferenciação

O *plugin* GeoKit permite tirar partido de bastantes funcionalidades para implementar em aplicações que necessitem de georreferenciação. Depois de instalado o *plugin* é necessário definir quais os modelos que serão georreferenciados. Para isso adiciona-se a linha seguinte em cada um:

```
acts_as_mappable :default_units => :kms, :distance_field_name => :distance
```

Por omissão isto implica que o modelo tenha as colunas `lat` e `lng` criadas. Com isto passamos a poder incluir no método `find` destes modelos parâmetros como origem e distância a determinadas coordenadas:

```
Object.find(:all, :origin => [37.792,-122.393], :within => 5)  
Object.find(:all, :origin => '100 Spear st, San Francisco, CA')
```

Existem algumas alternativas ao método `find` que permitem diferentes tipos de procura e podem ser consultadas na documentação do *plugin*. Este tipo de funcionalidade vai ser fundamental para a pesquisa com base na proximidade a pontos de interesse, também eles georreferenciados.

Para identificação das coordenadas de um dado quarto, dada uma determinada morada, é necessário recorrer a um *geocoder*. Esta funcionalidade está presente no *RoomBlick*, e utilizou-se o Google Maps. Para isso é necessário obter uma chave válida para utilização da API na nossa aplicação e alterar o ficheiro de configuração (`config/environment.rb`) de modo a definir esse chave, da seguinte forma:

```
GeoKit::Geocoders::google = '<chave>' if ENV['RAILS_ENV'] == "development"  
GeoKit::Geocoders::google = '<chave>' if ENV['RAILS_ENV'] == "production"
```

É importante perceber que esta chave não pode ser utilizada em IPs diferentes, e por essa razão se definem aqui duas chaves consoante o ambiente em que a aplicação corre. Estas chaves também são necessárias para apresentar o mapa com a localização do quarto. Para procurar as coordenadas é feita uma chamada a uma

função em Javascript disponibilizada pela API (`getLocations`). Esse procedimento foi encapsulado numa função mais genérica com tratamento de erros:

```
function getAddress(search) {
  geo.getLocations(search, function (result) {
    if (result.Status.code == G_GEO_SUCCESS) {
      var p = result.Placemark[0].Point.coordinates;
      var lat=p[1];
      var lng=p[0];

      // mostrar resultados no mapa se for caso disso
    }
    else {
      if (result.Status.code == G_GEO_TOO_MANY_QUERIES) {
        nextAddress--;
        delay++;
      }
      else { // tratar o erro }
    }
  });
}
```

Para a apresentação visual de um quarto com as suas coordenadas já guardadas na base de dados, é necessário utilizar uma série de funções da API do Google Maps em sequência, havendo muitos pontos de personalização. Aqui apresenta-se a implementação em traços gerais, havendo detalhes nas opções de configuração dos mapas que devem ser consultadas na API disponibilizada. Em primeiro lugar é necessária a inclusão de uma DIV no HTML onde vai figurar o mapa:

```
<div id="map_container">
  <div id="map" style="width:400px;height:400px;"></div>
</div>
```

Depois, nessa mesma vista, é necessário inicializar o mapa, que faz *callback* a uma função para possíveis operações:

```
<script type="text/javascript">
  google.load("maps", "2");
  google.setOnLoadCallback(showMap);
```

```
function showMap() {
    initialize_show(<%= "#{@ad.lat}, #{@ad.lng}" %>);
}
</script>
```

Depois de inicializado o mapa é então apresentado o quarto nas coordenadas correspondentes. Para isso utilizou-se a seguinte função em Javascript, que recebe as coordenadas de um quarto:

```
function initialize_show(lat, lng) {
    if (!map) {
        map = new google.maps.Map2(document.getElementById("map"));
        map.setCenter(new google.maps.LatLng(lat,lng), 16, G_HYBRID_MAP);
        // criar marca
        var point = new GLatLng(lat, lng);
        var marker = createMarker(point, map);
    }
    else { map.setCenter(new google.maps.LatLng(lat,lng), 16, G_HYBRID_MAP);}
}
```

A função `createMarker(point, map)` é usada para definir o tipo de marca que define a localização do quarto, que de uma forma simplificada pode-se resumir a:

```
function createMarker(point, map, type) {
    var baseIcon = new GIcon();
    baseIcon.iconSize = new GSize(32,32);
    baseIcon.shadowSize = new GSize(56,32);
    baseIcon.iconAnchor = new GPoint(16,32);
    baseIcon.infoWindowAnchor = new GPoint(16,0);
    var marker = new GMarker(point, {draggable:false,icon:roomIcon});
    map.addOverlay(marker);
    return marker;
}
```

## Detecção de IPs

Com o *plugin* GeoKit é também possível fazer a detecção automática de IPs, sendo útil, por exemplo, para apresentar a interface na língua adequada. A localização pode ser obtida da seguinte forma, onde <IP> seria o endereço IP do pedido:



```
location = IpGeocoder.geocode('<IP>')
```

É preciso ter em conta que este método nem sempre devolve uma localização, uma vez que pode não conseguir localizar o IP do utilizador. Para detecção automática terá de se alterar o controlador genérico da aplicação (`application.rb`) e adicionar as seguintes linhas logo no início:

```
geocode_ip_address  
before_filter :set_lang
```

O *helper* `geocode_ip_address` guarda automaticamente num *cookie* a localização do utilizador com base no endereço IP. A função `set_lang` tem de ser definida, e apenas selecciona a língua guardada ou a língua escolhida por omissão caso não seja possível aferir qual a localização do utilizador:

```
def set_lang  
  set_locale GeoKit::Default_lang if cookies[:geo_session].nil? or  
  cookies[:geo_session].empty?  
  set_locale cookies[:geo_session]  
end
```

## Exportação para KML

Fez parte do trabalho em georreferenciação a exportação dos dados georreferenciados para o Google Earth. Para isso teve de se registar um novo *Mime Type* nas configurações da aplicação (`config/initializers/mime_types.rb`):

```
Mime::Type.register "application/vnd.google-earth.kml+xml", :kml
```

Desta forma a aplicação reconhece e devolve KML, conseguindo construir vistas especiais quando é recebido um pedido para formato KML. Exemplos de pedidos válidos são:

```
http://exemplo.com/ads/index.kml  
http://exemplo.com/ads/492.kml
```

As respostas são construídas com base na biblioteca de manipulação de XML interna à *framework*, tendo em atenção a especificação mínima para uma formatação válida em KML. Assim, podemos construir as vistas KML para cada quarto, criando o ficheiro `room.kml.builder`. O próprio nome indica à aplicação Rails qual o tipo de

ficheiro devolvido e que sistema utilizar para o *render* (neste caso usa-se o *builder*, sistema de *render* de XML interno).

O exemplo seguinte é então a vista para um quarto com o nome acima indicado:

```
xml = Builder::XmlMarkup.new(:indent => 2)
xml.instruct! :xml
xml.kml("xmlns" => "http://earth.google.com/kml/2.2") do
xml.Document {
  xml.name("#{@ad.id}.kml")
  xml.description("RoomBlick")
  xml.Style( "id" => "highlight" ) {
    xml.IconStyle {
      xml.Icon {
        xml.href("http://maps.google.com/mapfiles/kml/pal2/icon20.png")
      }
    }
  }
  xml.Placemark {
    xml.name @ad.title
    xml.description image_tag(@ad.first_image.absrc) + "<br /><br />" +
    @ad.description +
      "<p>#{link_to 'Detalhes &raquo;', ad_path(@ad)}</p>"
    xml.styleUrl("#highlight")
    xml.Point {
      xml.coordinates("#{@ad.lng},#{@ad.lat},0")
    }
  }
}
end
```

# Anexo I

---

## Guia de utilização de Gettext Localization

Neste documento descrevem-se em linhas gerais os passos a seguir para conseguir obter internacionalização e localização com o *plugin Gettext Localization*, que foi utilizado e escolhido entre algumas alternativas disponíveis. O exemplo analisado é a tradução da aplicação *RoomBlick*, cuja língua é, por omissão, o inglês e que se quer adicionar suporte para português.

Para utilizar este *plugin* é necessária a instalação prévia da *gem "gettext"*:

```
sudo gem install gettext
```

É aconselhável fazer *freeze* desta *gem* para a nossa aplicação, de modo a poder fazer possíveis alterações em localizações já disponibilizadas que poderão não ir ao encontro dos formatos pretendidos. Essas traduções são feitas usando o mecanismo de ficheiros `.po` habitual quando se lida com traduções envolvendo a biblioteca *Gettext*.

É necessário então adicionar as seguintes linhas ao ficheiro `environment.rb` para carregar as bibliotecas necessárias:

```
require 'jcode'  
require 'gettext/rails'
```

Pode também definir-se, no mesmo ficheiro qual o local escolhido por omissão:

```
GettextLocalize::default_locale = pt_PT
```

Para definir um local é necessário escolher no controlador da aplicação (`application.rb`) um local válido, como por exemplo:

```
set_locale('de')
```

Essa selecção pode ser feita pelo utilizador, pelo que é necessário implementar um mecanismo de a alterar em *runtime* ao invés de estar definido no controlador. Assim,

constrói-se um *array* com as línguas disponíveis e depois inclui-se no *layout* da aplicação um método de selecção da língua desejada, por exemplo:

```
<% form_for :language, :url => "/langs/set", :html => {:id => 'lang_form'}
do |f| %>
  <%= f.select(:lang, @languages, {:selected => cookies[:geo_session]},
  {:onchange => "$('lang_form').submit()"}) %>
  <%= image_tag "#{GetText.locale.language}.png", :alt =>
  "#{GetText.locale.language}" %>
<% end %>
```

Como se pode ver, ao seleccionar-se a língua é chamado uma acção num controlador específico que muda a língua activa. No caso do *RoomBlick* essa acção pode ser simplificada na seguinte versão:

```
def set
  code = params[:language][:lang]
  cookies[:geo_session] = {
    :value => code,
    :expires => Time.now + 1.year,
    :path => '/'
  }
end
```

As *views* devem ser todas preparadas para aceitar conteúdos que mudam consoante a língua definida. Assim, apresentamos um excerto de uma *view* para que se perceba o que essa tarefa envolve:

```
<li>
  <label for="search_region"><%= Ad.human_attribute_name("region")
%></label>
  <%= collection_select(:search, :region, @regions, :id, :name) %>
</li>
<li>
  <label for="search_text"><%= _ "Type" %></label>
  <%= f.select(:tipo, {_("Rent") => 1, _("Share") => 0}) %>
</li>
<li>
  <%= link_to _("I need more search options"), search_path %>
</li>
```

```
<li>
  <%= f.submit _("Search"), :class => 'button', :disable_with =>
    _("searching&hellip;") %>
</li>
```

Todos os conteúdos a traduzir têm de estar dentro da função () *que pode ser abreviada usando a tag especial* `<%=_ %>` nas *views*. Exceptuam-se casos específicos como os atributos de um modelo. No excerto de uma *view* mostrado acima, a chamada a `Ad.human_attribute_name("region")` devolve “Região” se o local estiver definido para Portugal e a tradução tiver sido feita.

Para efectuar as traduções da aplicação, em primeiro lugar é necessário definir qual o nome do ficheiro, incluindo a seguinte linha no controlador da aplicação:

```
init_gettext "roomblick"
```

O ficheiro com sobre o qual são feitas as traduções pode ser gerado correndo a *rake task* seguinte:

```
rake gettext:updatepo
po/roomblick.pot
..... done.
```

Esta *task* vai extrair todas as ocorrências de textos a traduzir em todos os ficheiros da aplicação, bem como os nomes dos modelos e dos seus atributos. Para criar uma nova tradução basta copiar o ficheiro gerado para a directoria `po/pt_PT/` com o nome `roomblick.po`. A partir daí, se for corrida outra vez a *task* para actualização das entradas no ficheiro de tradução, os ficheiros criados dentro da directoria `po/` vão também ser actualizados.

Depois de editado um ficheiro de língua, pode então ser executada a seguinte *rake task*:

```
rake gettext:makemo
po/pt_PT/anuncios.po -> locale/pt_PT/LC_MESSAGES/anuncios.mo
```

Como se pode ver pelo *output*, isto vai gerar um novo ficheiro dentro da directoria `locale/` e é este que vai ser lido pela aplicação.

# Anexo J

---

## Pesquisa

### 1. Questionário

Aqui apresenta-se o questionário, disponibilizado na Internet, utilizado na tentativa de apurar quais os parâmetros considerados mais importantes no arrendamento de quartos em Portugal. Este questionário foi disponibilizado *on-line*, entre o dia 18 de Fevereiro de 2008 e dia 24 do mesmo mês. A formatação aqui apresentada não está de acordo com a original, em HTML:

No âmbito do estágio que estou a fazer surgiu a necessidade de elaborar este pequeno questionário para algumas tomadas de decisão. Para o preencher imagine que tem de arrendar um quarto em Portugal por alguma razão (se nunca teve de fazê-lo).

Nome: \_\_\_\_\_

Género: \_\_\_\_\_

Idade: \_\_\_\_\_ anos

Ocupação: \_\_\_\_\_

Para cada um dos parâmetros em baixo indique de 1 (**nada importante**) a 5 (**muito importante**) qual a importância que lhes atribua caso pretendesse alugar um quarto.

1. Valor da renda.
2. Localização do imóvel.
3. Partilha da casa com outros inquilinos.
4. Distância a pontos de interesse (bancos, universidades, metro, etc.).
5. Tipo de imóvel (moradia/apartamento).
6. Número de casas-de-banho.
7. Casa mobilada.
8. Televisão.
9. Quarto simples ou duplo.
10. Casa-de-banho privativa.
11. Acesso a cozinha.
12. Fogão.

13. Frigorífico.
14. Micro-ondas.
15. Máquina de lavar roupa.
16. Máquina de lavar loiça.
17. Televisão por Cabo.
18. Internet.
19. Elevador.
20. Presença/ausência de inquilinos fumadores.
21. Presença/ausência de inquilinos com animais de estimação.

## 2. Resultados

A amostra foi de 230 indivíduos, dos 18 aos 65 anos, de ambos os géneros e com diferentes ocupações, tendo-se abrangido uma variedade bastante grande de indivíduos. No entanto 63% dos inquiridos tinham menos de 35 anos, provavelmente por causa do meio onde o questionário foi disponibilizado e divulgado.

Na Figura 42 está a distribuição de pontuação média para cada um dos parâmetros numerados conforme o questionário apresentado em cima. Segundo esta distribuição tentou-se separar os parâmetros segundo a sua importância.

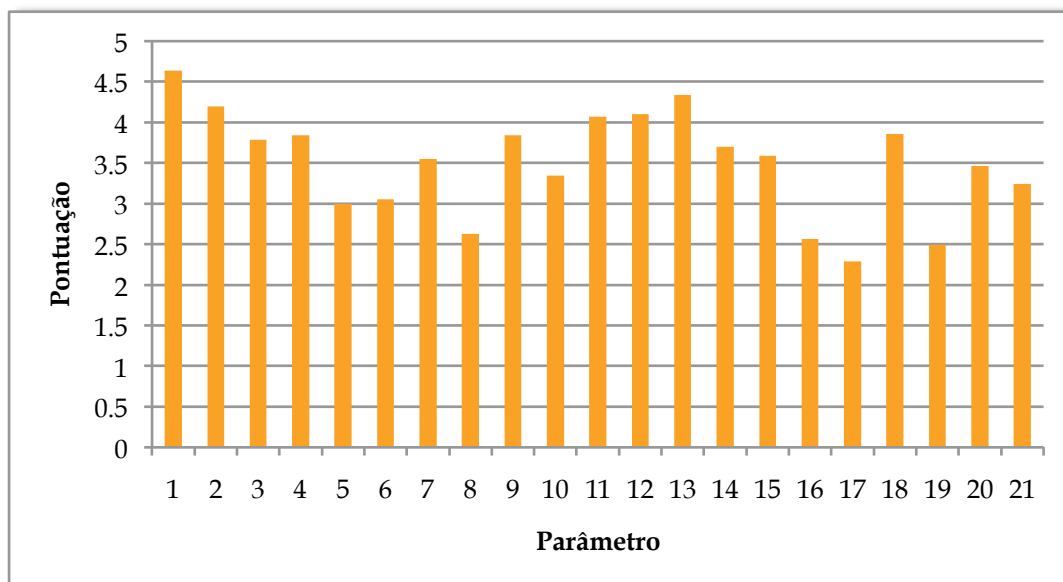


Figura 42 – Gráfico de distribuição de pontuação média para cada parâmetro.

Da análise do gráfico vemos que o primeiro parâmetro (valor da renda) foi considerado como o mais importante, tendo sido pontuado por uma grande maioria

acima de 4 valores. Segue-se a localização e depois parâmetros relacionados com equipamentos de cozinha. Esta distribuição foi tomada em conta para a organização da informação ao longo de toda a aplicação (por exemplo na ordem por que aparecem nos detalhes de um anúncio) para que vá ao encontro das expectativas dos utilizadores.

### 3. Implementação

Para uma pesquisa avançada, passar os parâmetros de pesquisa por um simples *request* GET ou POST ao controlador fazendo os filtros necessários depois na devolução de resultados não é uma aproximação muito eficaz e torna a funcionalidade difícil de manter. Se houver um número considerável de campos e a filtragem de resultados for complexa então faz sentido uma implementação diferente, tirando partido da filosofia REST.

Assim, primeiro cria-se um novo modelo `Search` que vai representar um *resource*, fornecendo os campos de pesquisa como atributos, as acções necessárias são `new` e `show`:

```
script/generate scaffold search category:integer ... new show
```

A pesquisa propriamente dita vai então ser efectuada directamente no modelo de pesquisa criado anteriormente. Cada atributo do modelo (parâmetro de pesquisa) vai ter um método para definir as condições e no fim essas condições são todas combinadas:

```
def ads
  @ads ||= find_ads
end

private

def find_ads
  Ad.find(:all, :conditions => conditions)
end

def category_conditions
  ["products.category_id = ?", category_id] unless category_id.blank?
```



```

end

# ... outras condições com base nos parametros

def conditions
  [conditions_clauses.join(' AND '), *conditions_options]
end

def conditions_clauses
  conditions_parts.map { |condition| condition.first }
end

def conditions_options
  conditions_parts.map { |condition| condition[1..-1] }.flatten
end

def conditions_parts
  private_methods(false).grep(/_conditions$/).map { |m| send(m) }.compact
end

```

Este método pode não ser adequado a situações onde é necessário fazer uma pesquisa por texto, mas na aplicação em questão resolve o problema de código pouco claro e difícil de manter. Para apresentar os resultados basta fazer *render* do resultado do método `ads` de uma instância deste modelo:

```
render :partial => @search.ads
```

Como estamos a criar um novo objecto sempre que é efectuada uma pesquisa pode ainda implementar-se facilmente uma associação do resultado de uma pesquisa (ou várias) a um utilizador o que pode ser muito vantajoso em algumas aplicações. Claro que este processo traz a desvantagem de se ocupar espaço em base de dados com pesquisas.

# Anexo K

---

## Guia de implementação de caching

### 1. Counter caching

Este tipo de *caching* é muito específico e a sua implementação é bastante fácil. Devem ser procuradas situações em que é chamado uma contagem de colecções associadas a um modelo, alterando-se a tabela desse modelo para suportar uma coluna extra com o número de itens. Por exemplo, para o número de mensagens associadas a um anúncio é criada a seguinte migração:

```
class AddAdCounterCache < ActiveRecord::Migration
  def self.up
    add_column :ads, :comments_count, :integer, :null => false, :default =>
    0

    Ad.reset_column_information
    Ad.find(:all).each do |a|
      Ad.update_counters a.id, :comments_count => a.comments.length
    end
  end

  def self.down
    remove_column :ads, :comments_count
  end
end
```

A migração é um pouco mais complexa do que o que habitualmente se vê em Rails porque é necessário fazer logo o *update* do número de comentários na coluna criada. Corrida a migração basta definir no modelo dos comentários um parâmetro adicional na relação:

```
belongs_to :ad, :counter_cache => true
```

A partir daqui podemos fazer a substituição de chamadas tipo `ad.comments.count` para `ad.comments_count`.

## 2. Page caching e Action caching

*Page caching* é o mecanismo mais rápido, de implementar. Basta adicionar uma linha, no controlador onde está incluída a acção correspondente ao URL chamado. Por exemplo, se o URL for `http://exemplo.com/users/1` então teremos de adicionar a seguinte linha no topo do controlador `users_controller`:

```
cached_page :show
```

Quando analisado o *log* da aplicação poderá então ser lida a seguinte linha, indicativa que nos acessos seguintes os pedidos nem vão passar pelo código da nossa aplicação:

```
Cached page: /users/1.html (0.00011)
```

*Action Caching* implementa-se de forma muito semelhante, uma vez que a única diferença é que a aplicação é sempre chamada e os filtros são corridos. Assim, no topo do nosso controlador podemos ter:

```
before_filter :authenticate # <--- Verifica autenticação  
cached_action :show
```

É preciso atentar à ordem das linhas, aparecendo sempre o `before_filter` antes do `cached_action`. Pode então ler-se no *log* da aplicação:

```
Cached fragment: exemplo.com/users/1 (0.00004)
```

E em pedidos seguintes:

```
Fragment read: exemplo.com/users/1 (0.00000)
```

## 3. Sweepers

A tarefa mais complexa na implementação dos dois tipos de *caching* atrás referidos é que necessitam de ser acompanhados de *sweepers* para expiração automática das cópias em disco caso seja necessário. Para isso, em primeiro lugar é necessário definir a localização desses *sweepers* no ficheiro `environment.rb`:

```
config.load_paths += %W( #{RAILS_ROOT}/app/sweepers )
```

Assim, podemos criar um novo *sweeper* nessa directoria, como por exemplo:

```

class UserSweeper < ActionController::Caching::Sweeper
  observe User # Modelo que queremos observar

  def after_update(user)
    expire_cache_for(user)
  end

  def after_destroy(user)
    expire_cache_for(user)
  end

  private
  def expire_cache_for(record)
    expire_page(:controller => 'users', :action => 'show', :id =>
record.id)
  end
end

```

Estas classes poderão observar vários modelos e a expiração pode envolver apagar diversos ficheiros. O exemplo em cima é para *Page caching*, mas, para *Action caching* basta substituir as chamadas à função `expire_page` por `expire_action`.

Para que o controlador tenha conhecimento de um *sweeper* é necessário adicionar a seguinte linha no topo do ficheiro do controlador:

```
cache_sweeper :user_sweeper, :only => [:update, :destroy]
```

## 4. Fragment caching

Para fazer *cache* de fragmentos das nossas *views* podem usar-se múltiplos blocos semelhantes a este:

```

<% cache "user_#{@user.id}_ad_list" do %>
  <%= render :partial => @ads %>
<% end %>

```

Aqui estamos a guardar em *cache* a listagem de anúncios de um utilizador com o nome `user_1_ad_list`. Da primeira vez que este bloco de código é chamado pode ler-se no *log* da aplicação:

```
Cache fragment miss: views/exemplo.com/users/user_1_ad_list (0.00151)
```

E nos seguintes:

```
Cache fragment hit: views/exemplo.com/users/user_1_ad_list (0.00021)
```

O problema com este tipo de mecanismo está na expiração da cópia guardada, que tem de ser apagada explicitamente com uma chamada à função `expire_fragment`. Por outro lado tem a vantagem de se poder guardar os fragmentos no disco, em memória ou em memcached. Para isso terá de se escolher uma destas alternativas, escrevendo-se essa linha no ficheiro `environment.rb` de

```
ActionController::Base.fragment_cache_store = :file_store, "/path"  
ActionController::Base.fragment_cache_store = :memory_store  
ActionController::Base.fragment_cache_store = :mem_cache_store
```

Cada uma das opções permite ainda parâmetros adicionais.

# Anexo L

---

## Guia de realização de testes

### 1. Test Unit

*Test Unit* é a biblioteca de testes utilizada por omissão em Rails. Não requerer por isso nenhuma instalação adicional. Os testes são criados na directoria `test/` com nomes do tipo `modelo_test.rb` ou `controlador_controller_test.rb` consoante se tratar de um teste unitário ou funcional.

Imaginando um modelo `User` podemos ter por exemplo os seguintes testes simples num ficheiro `user_test.rb`:

```
def test_should_create_user
  assert_difference 'User.count' do
    user = create_user
    assert !user.new_record?, "#{user.errors.full_messages.to_sentence}"
  end
end

def test_should_require_name
  assert_no_difference 'User.count' do
    u = create_user(:name => nil)
    assert u.errors.on(:name)
  end
end
```

Como se pode ver o modo de escrever testes não se assemelha aos blocos de código de uma aplicação, havendo por isso a necessidade de familiarização com a sua escrita e funções associadas. O número de testes pode ser tão extensivo quanto se achar necessário.

Os testes funcionais testam o funcionamento correcto dos controladores. Seguidamente apresenta-se um exemplo de teste para o controlador do modelo acima referido:

```
def test_should_get_index
```

```
get :index
assert_response :success
assert_not_nil assigns(:users)
end
```

Este teste de exemplo básico verifica se é devolvida a listagem de utilizadores quando se chama a acção `index`.

Existem *rake tasks* já incluídas na *framework* que permitem correr os testes unitários e funcionais. Seguidamente apresentam-se essas *tasks* com respectivo possível *output*.

Testes unitários:

```
rake test:units
Started
.F
Finished in 0.119914 seconds.

1) Failure:
test_should_require_name(UserTest)
<nil> is not true.

2 tests, 3 assertions, 1 failures, 0 errors
rake aborted!
(See full trace by running task with --trace)
```

O resultado aqui mostrado, com base nos testes acima referidos, indica que um dos testes passou mas outro não, uma vez que ao tentar criar-se um utilizador sem um campo obrigatório não foi devolvido um erro. Isto porque a *task* foi corrida **antes** da implementação dessa funcionalidade. O mesmo acontece quando se corre o teste funcional apresentado:

Testes funcionais:

```
rake test:functionals
Started
E
Finished in 0.100765 seconds.

1) Error:
test_should_get_index(UsersControllerTest):
```

```
ActionController::UnknownAction: No action responded to index

1 tests, 0 assertions, 0 failures, 1 errors
rake aborted!
(See full trace by running task with --trace)
```

Para simplificar a leitura das mensagens de erro pode ser instalada a *gem* “*redgreen*” que apresenta na consola os testes que passam a verde, os que falham a vermelho e os erros a amarelo, como se pode ver na Figura 43.



```
>startea
.....F.....
Finished in 0.447417 seconds.

1) Failure:
test_message_should_have_body(MessageTest)
  [./test/unit/message_test.rb:8:in `test_message_should_have_body'
   /Library/Ruby/Gems/1.8/gems/activesupport-2.0.2/lib/active_support/testing/default.rb:7:in `run']:
  nil is not true.

28 tests. 41 assertions. 1 failures. 0 errors
```

Figura 43 – Output de um teste unitário apenas com uma falha.

## 2. Rspec

Para utilização de Rspec primeiro é necessário instalar a *gem* com as bibliotecas de testes:

```
sudo gem install rspec
```

De seguida é necessário instalar dois *plugins* em conjunto para cobrir todas as funcionalidades da *framework*:

```
ruby script/plugin install
svn://rubyforge.org/var/svn/rspec/tags/CURRENT/rspec
ruby script/plugin install
svn://rubyforge.org/var/svn/rspec/tags/CURRENT/rspec_on_rails
```

É também necessário criar um esqueleto básico onde assentam os testes, que fornece alguns *helpers* específicos para Rails e cria ficheiros base, consoante a estrutura da aplicação existente:

```
ruby script/generate rspec
```

Os testes são então criados na directoria `spec/` da nossa aplicação. Está também presente um ficheiro de configuração - `spec.opts` onde se pode definir qual o



formato do *output*. Para imprimir o *output* na consola em *runtime* com recurso à coloração da gem "redgreen" esse ficheiro deve ter o seguinte conteúdo:

```
--colour
--format
progress
--loadby
mtime
```

Seguidamente apresentam-se os mesmos testes da secção anterior, reescritos com esta biblioteca. Assim, os testes unitários vêm:

```
context "A new User" do
  specify "should be valid with a full set of valid attributes" do
    lambda{
      user = create_user
      user.should_not be_new_record
    }.should change(User, :count).by(1)
  end

  specify "should be invalid without a name" do
    lambda{
      u = create_user(:name => nil)
      u.should have_at_least(1).errors_on(:name)
    }.should_not change(User, :count)
  end
end
```

E o mesmo teste funcional escreve-se da seguinte forma:

```
context "The Users controller" do
  fixtures :users
  controller_name :users

  specify "should return an index of users" do
    get :index
    response.should be_success
    User.should have(10).records
  end
end
```

Aqui também se usam *fixtures* para mais facilmente apurar o correcto comportamento do controlador. No capítulo seguinte é explicada esta funcionalidade.

### 3. Fixtures

De modo a conseguir-se fazer testes mais facilmente pode-se criar um conjunto de dados de teste, que podem ser carregados para a base de dados (versão de ambiente de teste). Desta forma não retira-se dos ficheiros de teste essa funcionalidade, de modo a maior clareza. Estes dados podem ao mesmo tempo ser usados por vários testes.

Os ficheiros com *fixtures* devem ser guardados em `test/fixtures/` com o mesmo nome do modelo a eles associado. A sua escrita é muito simples, em ficheiros `.yaml`, como se pode ver pelo exemplo seguinte, de criação de dois utilizadores:

```
quentin:
  name: Quentin
  surname: Tarantino
  role: admin
  created_at: <%= 5.days.ago.to_s :db %>
miguel:
  name: Miguel
  surname: Fernandes
  role: user
  created_at: <%= Time.now.to_s :db %>
```

É possível incluir blocos de código em Ruby e cada entrada fica definida por um nome, podendo ser referenciada por esse nome (por exemplo, no parâmetro *role*, como o Rails sabe fazer a associação, pode definir-se a permissão do utilizador com base no nome de uma outra *fixture*).

# Anexo M

---

## Usabilidade

Os testes de usabilidade foram realizados em duas sessões nos dias 17 e 18 de Maio de 2008 a oito inquiridos de idades compreendidas entre os 19 e 31 anos. Nenhum utilizador tinha tido até à data qualquer contacto com a aplicação. Na tarefa 5 simulou-se a activação de um anúncio, com a escrita da mensagem no telemóvel mas sem que fosse realmente enviada.

### 1. Questionário

Obrigado por ter aceite o convite a participar neste teste. Hoje vou pedir-lhe que sirva como avaliador deste sistema e complete algumas tarefas. O objectivo é perceber quão fácil é para si usá-lo. Todas as suas acções e reacções devem ser expressadas o mais possível, verbalmente, para que seja possível melhorar através do seu contributo.

**Não deve fazer perguntas à pessoa que o está a observar.** No entanto poderá haver interrupções para lhe perguntar porque efectuou determinadas operações.

Coisas a ter em mente:

- Isto não é um teste às suas capacidades por isso não tenha medo de errar.
- Não há resposta certa e errada para cada tarefa, tente cumpri-la como achar que é mais natural para si.
- Se estiver completamente perdido ou não compreender determinada tarefa avise a pessoa que o observa para que o esclareça
- Tente efectuar as tarefas **o mais rapidamente possível.**

Se tiver alguma pergunta pode fazê-la agora, antes de começar.

Divirta-se!

1. Aceda ao endereço <http://anuncios.mentesvirtuais.com> e crie uma nova conta de utilizador.

2. Com a nova conta criada faça autentique-se no sistema.
3. Edite o seu perfil e altere a sua fotografia.
4. Adicione um novo anúncio, como se fosse arrendar a sua casa. Forneça informação o mais próximo da realidade que conseguir.
5. Active o anúncio que acabou de criar para que fique disponível aos outros utilizadores.
6. Edite esse anúncio e associe-lhe, pelo menos, 3 imagens (se o anúncio já tiver 3 ou mais imagens então passe para o ponto seguinte).
7. Veja o anúncio como é apresentado aos outros utilizadores certificando-se que as imagens foram bem adicionadas.
8. Edite novamente esse anúncio e reordene as imagens nele contidas.
9. Adicione um outro anúncio à sua escolha e active-o.
10. Pesquise por um quarto em Coimbra, o mais próximo possível do Polo II.
11. Verifique se nesse quarto pode ter acesso a um frigorífico.
12. Confirme se a casa tem jardim nas traseiras com piscina.
13. Encontre agora um quarto próximo do Polo II mas com frigorífico.
14. Envie um pedido de arrendamento para o anunciante desse quarto.
15. Envie também uma mensagem privada ao anunciante perguntando se alguma vez teve problemas de baratas.
16. Veja os comentários colocados nesse anúncio e responda a um deles, à sua escolha. (o conteúdo da resposta é irrelevante)
17. Faça logout do sistema.
18. Imaginando que se esqueceu da sua *password*, tente recuperar o acesso.
19. Mude a língua em que lhe é apresentada a informação.
20. Verifique qual é o email de contacto dos responsáveis por esta aplicação.

## 2. Resultados

Na Tabela 4 estão registadas as observações gerais, fazendo-se o apanhado da experiência dos oito utilizadores, para cada passo do guião.

| Passo |  |
|-------|--|
| 1     | A maioria dos utilizadores conseguiram efectuar este passo sem problemas. O <i>link</i> de OpenID causou alguma estranheza havendo utilizadores que perdem alguns segundos a ler a caixa de diálogo. Três utilizadores não gostaram do facto de lhes ser pedido o telemóvel. |

|    |   |
|----|---|
| 2  | Todos os utilizadores conseguiram efectuar esta operação de forma rápida. Houve no entanto alguma estranheza por parte de dois dos inquiridos por não desaparecerem os links de registo depois de feito o login.  |
| 3  | Nenhum utilizador apresentou dificuldades no cumprimento da tarefa. A operação foi feita muito rapidamente e sem problemas.   |
| 4  | Aqui as dificuldades surgiram apenas na parte correspondente à localização. Apenas dois utilizadores leram as instruções de utilização do mapa e surgiram diversos problemas.   |
| 5  | Não houve dificuldade em encontrar o <i>link</i> correcto. A palavra 'roomblk' utilizada na mensagem foi no entanto uma dificuldade apontada na escrita da mensagem no telemóvel uma vez que não aparece no dicionário.   |
| 6  | Nenhum utilizador teve problemas no cumprimento desta tarefa.   |
| 7  | Nenhum utilizador teve problemas no cumprimento desta tarefa. No entanto, há que referir que apenas um verificou se o anúncio estava disponível não pelo <i>link</i> do anúncio na área privada mas sim pela listagem de quartos.   |
| 8  | Aqui foi descoberto um problema de usabilidade grave. Sete utilizadores não conseguiram editar rapidamente o anúncio ou por não encontrarem nenhum <i>link</i> directo na página de detalhes desse anúncio ou por perderem algum tempo voltando à página inicial novamente. |
| 9  | Depois da primeira experiência não houve reincidências em problemas anteriores. Todos os utilizadores cumpriram esta tarefa correctamente.  |
| 10 | Todos os utilizadores cumpriram a tarefa sem problemas mas houve alguma estranheza por parte de um deles quanto ao facto de não se poder seleccionar outro tipo de ponto de interesse que não os disponibilizados.  |
| 11 | Este foi outro ponto onde houve bastantes dificuldades. A forma como esta informação está disponibilizada não é eficaz. Seis dos oito utilizadores tiveram problemas em encontrar a informação.   |
| 12 | Cinco dos inquiridos não souberam encontrar esta informação de forma eficiente, entre os quais dois acharam mesmo que seria uma falha da aplicação não se lembrando de verificar o mapa da localização do quarto. Isto pode estar ligado ao problema do ponto anterior.     |
| 13 | A tarefa foi cumprida sem problemas por todos os utilizadores.  |
| 14 | Apesar de todos terem cumprido a tarefa com bastante celeridade houve um utilizador que perdeu mais tempo do que seria de esperar à procura do link.  |
| 15 | Todos os utilizadores cumpriram a tarefa sem qualquer problema.   |
| 16 | Nenhum utilizador teve dificuldades em encontrar os comentários. Houve no entanto alguma confusão por se criarem duas caixas de texto.  |
| 17 | Nenhum utilizador teve dificuldades em executar a tarefa.   |
| 18 | Nenhum utilizador teve dificuldades em executar a tarefa.   |
| 19 | Nenhum utilizador teve dificuldades em executar a tarefa.   |
| 20 | O contacto é assumido como estando na parte inferior do <i>layout</i> e é encontrado de imediato.   |

Tabela 4 – Tabela de resultados das sessões de teste de usabilidade.

No final de cada sessão de teste foi pedida uma apreciação global do sistema e pontos que tivessem achado menos bem conseguidos. O modo pouco eficiente de como são apresentadas as comodidades existentes foi um dos pontos apontados como mais desagradáveis. A confusão ao inserir as coordenadas de um quarto também teve maior destaque pela negativa uma vez que o erro de submeter incorrectamente o anúncio obriga a ter de o editar novamente.

Na Tabela 5 agrupam-se as observações resultantes da realização dos testes de usabilidade indicando possíveis resoluções ou reacções de utilizadores e suas sugestões. A coluna “Num” quantifica o número de utilizadores que partilharam da mesma dificuldade.

| Passo | Dificuldade   | Num | Possível resolução  |
|-------|---|-----|---|
| 1     | Link de OpenID causa confusão e faz perder algum tempo                        | 2   | Ocultar essa informação partindo do princípio que os utilizadores que usam OpenID sabem identificar o icon.   |
| 1     | Desagrado com pedido de telemóvel   | 3   | Adicionar explicação quanto ao propósito deste pedido ou tornar o parâmetro como opcional.  |
| 1     | Calendário aparecer mal se clica no campo de texto                            | 1   | Reformular o modo como é apresentado o campo ou permitir que o utilizador escreva a data à mão informando-o do formato correcto.  |
| 2     | Estranheza com a permanência de links de registo                              | 2   | Remover os links para registo podendo ser substituídos por outros. Foi sugerido que houvesse um link directo no menu para os meus anúncios.   |
| 4     | Não foi feita a leitura de instruções de manuseamento do mapa.                | 6   | Simplificar o processo tornando-o mais natural ou, se necessário, mover as instruções para um sítio com mais destaque.  |
| 4     | Ao inserir-se a morada carregou-se no <i>Enter</i> fazendo submeter o anúncio | 4   | Não fazer submeter o anúncio pressionando <i>Enter</i> ou, segundo sugestão de um utilizador, tentar mapear automaticamente, de modo transparente, consoante a morada escrita. Este é um erro que os utilizadores consideraram muito desagradável de recuperar. |
| 4     | O link de mapear foi clicado antes de escrita a morada                        | 2   | Reformular o modo como é feito este mapeamento, como referido acima.  |
| 5     | Desagrado com o facto de ser uma  | 4   | Simplificar a keyword.  |

|    |  |   |   |
|----|--|---|---|
|    | mensagem de escrita não muito agradável  |   |   |
| 5  | Engano na escrita da mensagem. É um erro grave uma vez que o utilizador iria perder dinheiro e o anúncio não seria activado. | 1 | O carácter 'í' foi confundido com o carácter 'l'. Poder-se-ia apresentar o texto na página com texto maior ou com outra <i>font</i> .   |
| 6  | Necessidade de o <i>link</i> de adicionar mais uma imagem ir descendo acompanhando os campos e não ficar no cimo.            | 1 | Fazer com que o link acompanhe o aumento de campos.   |
| 8  | É perdido muito tempo na tentativa de encontrar um <i>link</i> directo para editar o anúncio                                 | 7 | Adicionar uma forma fácil de editar o anúncio, por exemplo adicionando um <i>link</i> directo na página de detalhes quando é o próprio anunciante que a visita.   |
| 11 | Não foi encontrada informação necessária levando a muito tempo perdido e alguma exasperação                                  | 6 | A listagem das comodidades de um quarto ocultando as que não estão presentes relevou-se uma técnica pouco eficiente e confusa. Em alternativa poderia ser adicionada uma lista com todas as comodidades existentes indicando quais as que o quarto tinha acesso ou não. |
| 12 | Dificuldade em encontrar a informação levando a muito tempo perdido e alguma exasperação                                     | 5 | Talvez dependente da resolução do problema anterior. Quanto ao caso concreto de verificar aspectos físicos da casa, isso poderia ter sido evitado com fotografias da casa.  |
| 14 | Alguma dificuldade em encontrar o <i>link</i>  | 1 | Dar mais ênfase ao <i>link</i> .  |
| 16 | Confusão pela presença de caixas de texto múltiplas  | 2 | Ocultar a caixa de texto apresentada por defeito no fundo podendo funcionar de modo semelhante à utilizada no <i>link</i> de <i>reply</i> .   |

Tabela 5 – Dificuldades encontradas em cada tarefa do guião de usabilidade e possíveis resoluções.