FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

WaaS: Wiki as a Service

André Filipe Monteiro Lamelas da Silva

Relatório de Projecto realizado no Âmbito do
Mestrado Integrado em Engenharia Informática e de Computação

Orientador: Ademar Manuel Teixeira Aguiar (Ph.D)

Julho de 2008

WaaS: Wiki as a Service

André Filipe Monteiro Lamelas da Silva

Relatório de Projecto realizado no Âmbito do
Mestrado Integrado em Engenharia Informática e de Computação

Aprovado em provas públicas pelo júri:

Presidente: Ana Paula Cunha da Rocha (Professora Doutora)

Arguente: António Rito Silva (Professor Doutor)

Vogal: Ademar Manuel Teixeira Aguiar (Professor Doutor)

7 de Julho de 2008

Resumo

Um *wiki* é uma página *web* editável por qualquer utilizador autorizado, escrita numa linguagem pré-definida que depois é traduzida em HTML. Apesar de simples, as implicações deste conceito nas tecnologias *web* que possuímos hoje em dia são imensas. Assistimos nos últimos anos a uma mudança no paradigma em relação aos conteúdos na World Wide Web. Dantes, os conteúdos estavam lá, produzidos por alguém cuja função era a de produzir conteúdos; hoje em dia, na chamada Web2.0, os conteúdos são também produzidos por nós.

Para permitir que essas faculdades de criação de conteúdos estejam acessíveis a todos é necessária uma progressiva simplificação das ferramentas de criação de conteúdos *web*. É neste âmbito que aparecem os requisitos de usabilidade que hoje são fundamentais no desenvolvimento de aplicações *web*.

Neste relatório descreve-se o processo de desenvolvimento da aplicação Gruki, um *wiki* construído com o objectivo de permitir a fácil criação de conteúdos por todos, para todos. A utilização do editor WYSIWYG de conteúdos *web* Wedit, desenvolvido com a **usabilidade** em mente, permite que o Gruki adquira para si essas qualidades que o tornam uma aplicação simples e fácil de usar.

O Gruki pretende igualmente simplificar a utilização de *wikis*, permitindo a criação de um *wiki* através de um simples registo. Cada *wiki* do Gruki será denominado por "espaço", sendo que cada espaço tem associado um conjunto de páginas sobre o qual é aplicado um determinado conjunto de permissões. Cada espaço possui igualmente um conjunto de utilizadores associado que sobre este têm permissões especiais.

De forma a agilizar o processo de desenvolvimento do Gruki e com o intuito de explorar os métodos ágeis de desenvolvimento de *software*, vai ser usada a *framework* Ruby on Rails, uma *framework* para o desenvolvimento de aplicações *web* que é altamente orientada para os métodos ágeis de desenvolvimento de *software*. Estes métodos são baseados numa filosofia de que o desenvolvimento de *software* deve ser um processo ágil e iterativo, capaz de se adaptar facilmente ás mudanças.

Assim, este projecto não só irá permitir uma aprendizagem ao nível das metodologias ágeis de desenvolvimento de *software* como também irá ser uma aplicação *web* de elevado valor para a partilha colaborativa de informação motivada pela sua natureza *wiki*.

Abstract

A wiki is a webpage that can be edited by any authorized user, written in a pre-defined language that is afterwards converted in HTML. Although simple, the implications of this concept on the web technologies we have nowadays are enormous. We assisted through the last years to a paradigm shift concerning the World Wide Web contents. Before, the contents were there, produced by someone whose function was exactly to produce those contents; today, on the so called Web2.0, the contents are also created by us.

To allow those content creation functionalities to be accessible to everyone it is necessary to do a progressive simplification of the web content creation tools. This is the scope where usability requirements appear, being nowadays fundamental on the web applications development.

On this paper it's described the development process of the Gruki application, a wiki built with the purpose of allowing the easy creation of contents for by everybody, for everybody. The WYSIWYG web content editor Wedit, built around the **usability**, allows Gruki to acquire to itself that simplicity and ease of use.

Gruki also wants to simplify the use of wikis by allowing the the creation of a wiki using just a simple register form. Each wiki on Gruki will be called a "space", being each space associated with a group of pages that share a certain set of permissions. Each space has also a group of associated users that have special privileges in the space.

To allow an agilization of the Gruki development process and with the objective of exploring the agile methods of software development, the framework Ruby on Rails is going to be used. This is a framework for web applications development and it's highly oriented for the agile methods of software development. This methods are based in a philosophy that states that software development should be and agile and iterative process, making it able to adapt easily to change.

This way, this project not only will allow learning on the agile methodologies of software development but will also result in a web application with high value for collaborative information sharing motivated by it's wiki nature.

Agradecimentos

Este projecto é resultado de várias semanas de trabalho individual mas, como é sabido, nada é verdadeiramente feito sozinho.

Queria agradecer desde logo ao Engenheiro José Bonnet da PT Inovação cujas ideias e visão tornaram o Gruki num desafio que agarrei com vontade e dedicação. Para ele, o meu muito obrigado.

Gostaria igualmente de agradecer ao Professor Ademar Aguiar da FEUP pela orientação concedida em termos académicos e por todo o conhecimento relacionado com *wikis* que me forneceu.

Será igualmente justo agradecer à Cláudia Branco, colega estagiária da PT Inovação, com quem a troca ocasional de ideias sobre o Ruby on Rails resultou sempre muito frutífera.

Apesar de não terem ajudado directamente, um obrigado para o Miguel Biscaia e o Claúdio Lobo da PT Inovação, colegas "de carteira"cujo conhecimento e experiência contribuiram para um crescimento pessoal adicional fora de todo este projecto.

Igualmente importante é o agradecimento à minha subidíssima amiga Vanessa Costa, por ter feito a correcção linguística deste relatório.

Obrigado ainda à minha família e amigos por tornarem a batalha de todos os dias mais fácil de ganhar.

Quero ainda agradecer à Geek Tribe por ser tantas vezes o /dev/null das frustrações do dia-a-dia e local de agradável convívio, troca de ideias e de *spam*.

Por fim, um muito obrigado a John Coltrane, Bill Evans e Toumani Diabaté pela música magnífica e inspiradora que produziram e que, de uma forma ou outra, foram a banda sonora do desenvolvimento do Gruki.

André Filipe Monteiro Lamelas da Silva

Conteúdo

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

Abreviaturas e Símbolos

AJAX	Asynchronous Javascript And XML
CMS	Content Management System
CRUD	Create Update Read Delete
CSS	Cascade Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
MVC	Model-View-Controller
REST	Representational State Transfer
RSS	Really Simple Syndication
SGBD	Sistema de Gestão de Bases de Dados
XML	eXtensible Markup Language
WYSIWYG	What You See Is What You Get

Capítulo 1

Introdução

1.1 Contexto/Enquadramento

Quando, em 1995, Ward Cunningham criou a primeira *wiki* para servir como base para a troca de experiências no Portland Pattern Repository, certamente que não imaginaria o sucesso que o conceito viria a ter. Anos depois, o conceito de *wiki* vulgarizou-se e as suas aplicações são hoje imensas. As *wiki*s são usadas um pouco por todo o mundo com aplicações que vão desde a enciclopédia *online* (como a Wikipedia, que é o exemplo mais comum) ao suporte de processos de desenvolvimento de *software* (como no caso da PT Inovação). Por este motivo, há, por parte da PT Inovação, um interesse nas *wiki*s como ferramentas de suporte ao trabalho colaborativo.

A PT Inovação é uma empresa nascida em 1999 e que se afirma como "a âncora tecnológica do Grupo Portugal Telecom". Contando com cerca de 500 trabalhadores, a sua sede está localizada em Aveiro mas tem igualmente um Pólo no Porto e outro em Lisboa bem como uma divisão no Brasil.

Os produtos da PT Inovação estão em vários países do mundo como o Brasil, Cabo Verde, Angola, Botswana e até Timor. O seu volume de vendas em 2007 foi de cerca de 80 milhões de Euros anuais.[Ino08]

Os seus principais produtos são o NGIN, o Netb@nd e o NOSSIS. O NGIN é uma plataforma de serviços de rede inteligente que conta com mais de 70 milhões de utilizadores em todo o mundo. O Netb@and é uma solução integrada de rede de acesso e o NOSSIS é uma solução de suporte à operação e gestão de redes nos âmbitos da gestão, provisão, recursos e supervisão.

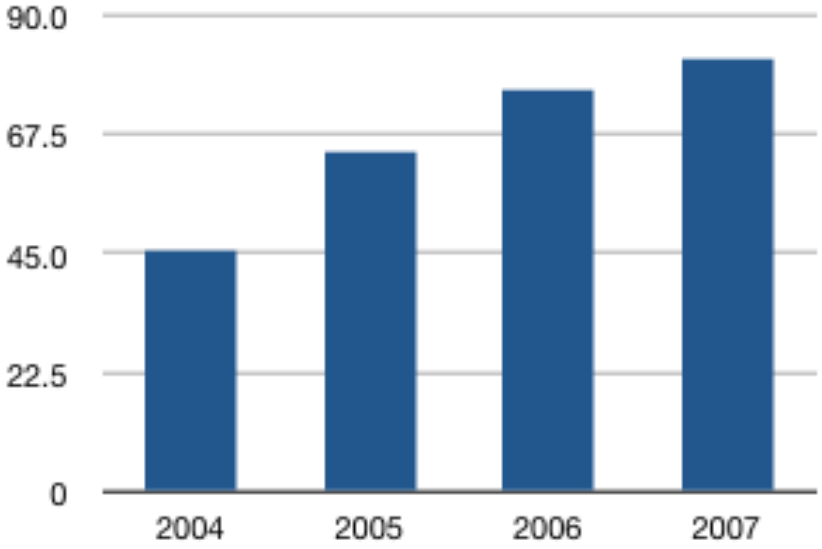

Figura 1.1: Volume de negócios da PT Inovação (em Milhões de Euros)

Apesar de o estágio ser realizado na PT Inovação, a empresa à qual estou associado primeiramente é a Inova-Ria.

A Inova-Ria é uma associação de empresas para uma rede de Inovação situada na zona de Aveiro e que conta com cerca de 50 empresas, entre as quais a PT Inovação. Os seus objectivos são criar e consolidar um cada vez maior número de empresas dos sectores das telecomunicações e tecnologias de informação, bem como promover a inovação e a cooperação empresarial, nomeadamente aos níveis de Investigação e Desenvolvimento, Formação, Marketing e Internacionalização. Convém juntar a isto o interesse em reforçar a posição da zona de Aveiro ao nível nacional e internacional na área das telecomunicações

1.2 Projecto

A aplicação a implementar será demoninada por Gruki, nome deriva da junção das palavras "grupo"e *"wiki"*. Durante grande parte do período de desenvolvimento da aplicação esta teve o nome de WikiSpaces, nome que foi mudado para impedir confusão com a aplicação Wikispaces já existente.[1]

Pretende-se com o Gruki implementar uma *wiki* recorrendo à *framework* Ruby on Rails. Esta *wiki* terá de ter as funcionalidades básicas de uma *wiki*: a criação e edição de

[1] Esta aplicação encontra-se em http://www.wikispaces.com

uma página e o agrupamento de páginas em "espaços". Todas as páginas de um espaço irão partilhar um conjunto de permissões que irão limitar o seu acesso.

O Gruki deverá ainda fornecer a possibilidade de se verificar o histórico de uma página, acrescendo à possibilidade de visionamento das várias versões anteriores da página a possibilidade de retornar a uma dessas mesmas versões.

Deverá ainda ser o usado o editor WYSIWYG Wedit, desenvolvido por um grupo de alunos da FEUP, na disciplina de Laboratório de Gestão de Projectos, no ano lectivo de 2006/2007. Este editor irá permitir uma maior simplicidade ao nível da utilização da aplicação, facilitando o seu uso por pessoas com menores conhecimentos de informática e sem familiaridade com as normais sintaxes *wiki*.

1.3 Motivação e Objectivos

O crescente número de utilizadores da *web* e a crescente utilização de novos serviços obriga a que as aplicações *web* tenham então de ser desenvolvidas segundo dois princípios essenciais para que estas estejam acessíveis a um maior número de pessoas com a maior das funcionalidades e constante evolução. Por um lado, há uma necessidade de as aplicações serem desenvolvidas de forma iterativa, sendo adicionadas funcionalidades em cada iteração e, por outro lado, requer-se aplicações de elevados padrões de usabilidade, de forma a permitirem uma maior facilidade no seu uso. A iteratividade permite que o produto seja testado em ambiente de utilização desde a primeira versão desenvolvida. Isto leva a um maior número de testes efectuados, levando assim à descoberta de erros que de outra forma só seriam descobertos em fases posteriores. Estes erros tanto podem ser *bugs* da aplicação como questões da simplicidade de utilização que não tinha sido previamente endereçadas.

As necessidades de agilidade remetem-nos para o Agile Manifesto, escrito em 2001 por um conjunto de autores (como o já referido Ward Cunningham). Este manifesto advoga um novo processo para o desenvolvimento de *software*, baseando-se num conjunto de princípios que defendem a agilidade do processo de desenvolvimento como meio de alcançar os requisitos pretendidos. Esta agilidade é conseguida através de uma grande ligação entre a equipa de desenvolvimento e o cliente, o que permite uma rápida entrega de versões incrementalmente mais funcionais, o que, por sua vez, consente uma validação/alteração dos requisitos à medida que a aplicação é desenvolvida.

A *framework* Ruby on Rails, usada como base para este projecto, foi desenvolvida tendo em mente os princípios do Agile Manifesto, o que faz dela um óptimo princípio para adquirir prática no uso de metodologias ágeis de desenvolvimento de *software*. Nos dias que correm, a velocidade do aparecimento de novas aplicações web faz com que a seu desenvolvimento tenha de ser rápido e ágil para conseguir acompanhar todos os novos

requisitos dos utilizadores, bem como corresponder à necessidade de inovação numa área sempre em mudança.

Por outro lado, as necessidades de usabilidade prendem-se com a facilidade com que um utilizador atinge o resultado por si desejado utilizando a aplicação em questão. A usabilidade é um assunto da ordem do dia pois, com o crescimento da *web*, cada vez mais pessoas com diferentes graus de conhecimento informático a usam, obrigando assim a uma progressiva simplificação dos padrões de utilização.

O desenvolvimento do Gruki é, então, útil como tubo de ensaio para métodos de desenvolvimento ágil de aplicações usáveis, o que o torna uma aplicação moderna e muito dentro do espírito da *web* que queremos para o futuro. Sendo uma *wiki*, o Gruki é, além de um caso de estudo do processo de desenvolvimento ágil, uma ferramenta de trabalho colaborativo, o que potencia o trabalho de equipa e o suporte a processos de desenvolvimento.

1.4 Estrutura da Dissertação

Neste relatório é descrito todo o processo de desenvolvimento do projecto Gruki, desde a sua definição de requisitos até à descrição da solução encontrada, bem como a sua contextualização e posterior análise de resultados.

- no **Capítulo 2** é definido com detalhe o problema apresentado, fazendo-se um levantamento de requisitos funcionais e não funcionais.
- no **Capítulo 3** é dedicado a uma revisão tecnológica/estado da arte em que farei uma análise exaustiva de soluções similares à que se pretende desenvolver. Analogamente, é feita uma abordagem aos principais conceitos relacionados com o projecto, como por exemplo o conceito de *wiki*, e às tecnologias a usar, com foco no RubyOnRails.
- no **Capítulo 4** descreve-se a solução implementada, sendo demonstrados a sua arquitectura e detalhes de implementação.
- no **Capítulo 5**, consistindo numa demonstração da aplicação, no qual se inclui ainda o Manual de Utilização da mesma.
- por fim, no **Capítulo 6** expõem-se as conclusões obtidas e efectua-se uma análise crítica da aplicação desenvolvida.
- no final do relatório estão as **referências** utilizadas.

Capítulo 2

Definição do Problema

O projecto que descrevo neste relatório surgiu para que a PT Inovação atingisse os seguintes objectivos:

- ***know-how* na *framework* Ruby on Rails:** sendo conhecida a agilidade com que se desenvolvem aplicações que envolvam da base de dados aos interfaces gráficos de utilizador baseados em *web* usando esta *framework*, a PT Inovação pretende conhecer melhor as restrições que se poderão colocar usando esta *framework* neste tipo de projectos;

- **reutilização do WEdit:** há igualmente um interesse em usar a aplicação Wedit, um editor WYSIWYG (What You See Is What You Get) desenvolvido no ano lectivo de 2006/2007 por um grupo de alunos da FEUP (Faculdade de Engenharia da Universidade do Porto) no âmbito da disciplina de Laboratório de Gestão de Projectos. Este editor foi desenvolvido tendo como principal objectivo a criação de um editor de conteúdos *web* de grande simplicidade e acessibilidade com o intuito de permitir a sua utilização pelo maior número possível de pessoas. Com a utilização deste editor, o Gruki tornar-se-á uma aplicação igualmente usável e acessível.

- **alojamento de *wikis*:** um outro objectivo quer seria desejável é o da possível comercialização dum serviço de alojamento de *wikis* se a aplicação desenvolvida atingir um estado em que seja fácil dar esse passo extra e se encontre na PT uma empresa patrocinadora desse lançamento.

Nesta secção iremos detalhar os requisitos associados ao Gruki. A especificação de requisitos é uma fase crucial no processo de desenvolvimento de *software*. A correcta especificação de requisitos permite que a aplicação a desenvolver fique o mais próximo possível do desejado.

Dentro da especificação dos requisitos do Gruki há que distinguir entre dois tipos de requisitos. Por um lado os requisitos funcionais referem-se ás funcionalidades do Gruki, isto é, aquilo que o Gruki faz ou permite ao utilizador fazer. Os requisitos não funcionais referem-se a questões que não têm directamente a ver com o que o Gruki faz mas estão relacionados com questões como a necessidade de certos valores de performances ou o cumprimento de standards de acessibilidade.

2.1 Requisitos Funcionais

Na sua essência o Gruki é uma *wiki* e como tal os seus requisitos funcionais são relativamente simples. O que distingue o Gruki das normais *wikis* é a existência de "espaços", conjuntos de páginas partilhadas por utilizadores que sobre elas têm permissões comuns. Este conceito de espaços permite, por exemplo, que vários utilizadores possam trabalhar colaborativamente em vários projectos separados entre si sem que os utilizadores não associados ao projecto lhe acedam. Outro aspecto que faz igualmente o Gruki diferente de outras aplicações já existentes é a utilização de um editor WYSIWYG para a edição de conteúdos.

Para facilitar o processo de levantamento de requisitos vou utilizar um diagrama de **casos de uso** de forma a tornar mais clara a sua visualização e seguidamente irei aprofundar cada uma das funcionalidades, explicando a sua utilidade e a sua pertinência para a qualidade da aplicação a desenvolver (ver figura 2.1).

Os papeis dos três actores presentes no diagrama são:

- o Visitante é o utilizador não registado
- o Utilizador é o utilizador registado e autenticado no sitema, pertencente ao espaço selecionado
- o Administrador de um Espaço é o criador de um espaço (o que lhe confere permissões administrativas)

O Visitante tem como possibilidades o **registo** na aplicação, fornecendo as informações que lhe forem pedidas. Após esse registo o Visitante pode agora **autenticar-se** na aplicação passando a ser um Utilizador. Existem três casos de uso que são cruciais para a lógica do Gruki e que estão acessíveis ao Visitante. Estes casos de uso são os de **criação**, **edição** e **visualização** de uma página. Convém contudo referir que estes casos de uso estão sujeitos ao sistema de permissões do Espaço em que as páginas em questão estejam inseridas. Existem três tipos de permissões que passamos a definir e que ajudam a entender como funcionam estes casos de uso:

- Espaço Público - quer Visitantes, quer Utilizadores podem ver, criar e editar páginas

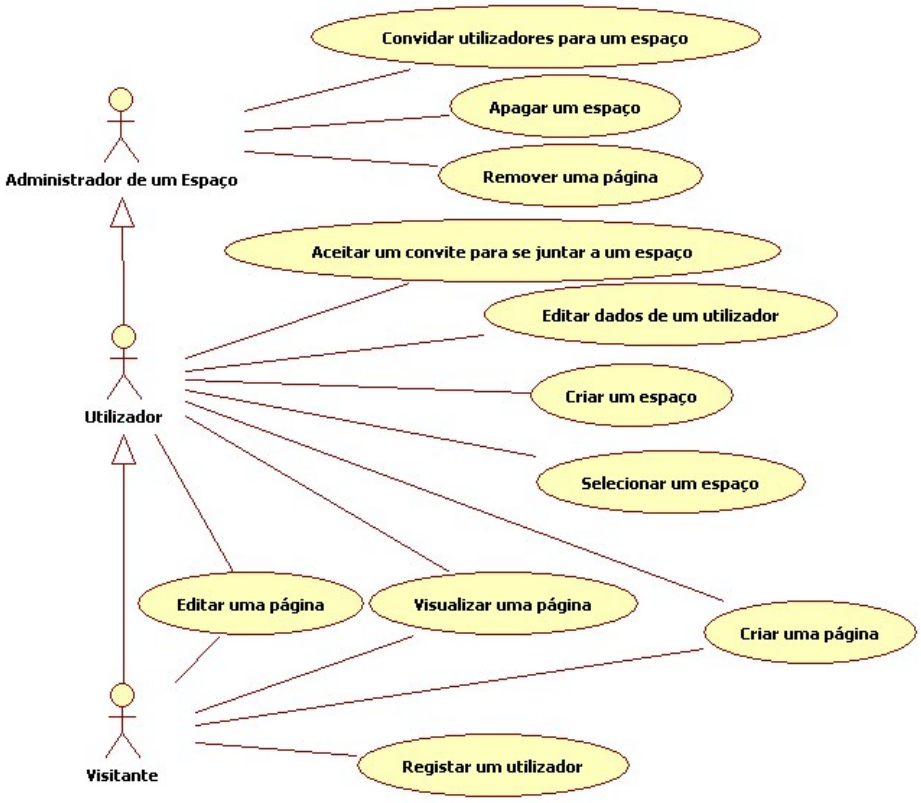

Figura 2.1: Casos de Uso do Gruki

- Espaço Aberto - as páginas deste espaço podem ser vistas por Visitantes e Utilizadores mas apenas Utilizadores daquele espaço podem editar e criar páginas.

- Espaço Fechado - apenas Utilizadores pertencentes ao espaço selecionado podem ver, editar e criar páginas

Existe ainda um tipo especial de permissão que é o de Administrador de um Espaço. Um Administrador de um Espaço é o utilizador que o criou e, como tal, é o único que pode **apagar** coisas no Espaço definitivamente, sejam páginas ou o próprio Espaço. O Administrador pode ainda **convidar** novos utilizadores a juntarem-se ao espaço.

Um Utilizador registado pode criar um espaço novo, para o qual poderá escolher quais as permissões que este terá. O Utilizador poderá igualmente **alterar** as suas definições pessoais como o email ou a password com que efectuou o registo. Seria interessante também a existência de um ***dashboard*** de actividade, onde se mostrariam quais as últimas páginas editadas nos espaços a que um utilizador pertence.

Dentro das funcionalidades das páginas será também relevante a possibilidade de ver o **histórico** de uma página e de efectuar o ***rollback*** para uma versão anterior. Igualmente

interessante seria ver uma lista das páginas para as quais a página actual possui uma **hiperligação**.

Ao nível dos espaços, as funcionalidades mais relevantes são as de **criar** um espaço e definir permissões de acesso, sendo que isto pode ser definido no momento da criação do espaço e **alterado** posteriormente. Dentro de cada espaço também é útil poder aceder a uma listagem, quer de **utilizadores** membros de um espaço, quer de **páginas** pertencentes a um espaço. Uma funcionalidade que é útil por uma questão de gestão do espaço é uma página ***orfã***, ou seja, uma página para a qual nenhuma outra página faz uma hiperligação.

Vamos esquematizar estes requisitos funcionais na forma tabelar, prioritizando-os:

Tabela 2.1: Tabela de especificação e ordenação de requisitos

Número	Requisito	Prioridade
1	Criar página	Alta
2	Editar página	Alta
3	Visualizar página	Alta
4	Criar espaço com conjunto de permissões	Alta
5	Alterar permissões de um espaço	Alta
6	Registar utilizadores	Alta
7	Autenticação de utilizadores	Alta
8	Convidar utilizadores para um espaço	Alta
9	Listar utilizadores de um espaço	Média
10	Listar páginas de um espaço	Média
11	Remover um espaço	Média
12	Remover uma página	Média
13	Ver histórico de uma página	Média
14	Fazer ”rollback” de uma página	Média
15	Dashboard de actividade	Baixa
16	Listar páginas orfãs	Baixa

Os requisitos de prioridade Alta são aquelas cuja implementação é essencial para a concretização do projecto e sem os quais o projecto fica seriamente reduzido em termos de funcionalidade. A prioridade Média é associada a requisitos cuja implementação seria altamente desejável mas que não afectam gravemente o funcionamento da aplicação. Por fim, os requisitos de prioridade Baixa são os requisitos secundários que, apesar de adicionarem valor à aplicação, não são fundamentais.

2.2 Requisitos Não Funcionais

Os requisitos não funcionais são requisitos que, não estando directamente ligados às funcionalidades que o Gruki é suposto conter, estão relacionados com dependências e requistos externos à aplicação. Nesta secção iremos descrever esses requisitos dentro de várias áreas. Primeiramente iremos referir os requisitos ao nível das interfaces externas e posteriormente os requisitos em termos de usabilidade, confiabilidade, disponibilidade, portabilidade, escalabilidade e performance.

2.2.1 Requisitos de interfaces externas

Nesta secção explico quais as interfaces externas que são necessárias para que o Gruki seja usado. Foram excluídas desta lista as interfaces de ”hardware” uma vez que devido à portabilidade da plataforma sobre o qual foi implementado, o Gruki não requer nenhum ”hardware” específico.

Interfaces com o utilizador

Sendo uma aplicação ”web”, o Gruki deverá correr num ”browser”. Devido a limitações motivadas pelo editor WYSIWYG utilizado a edição de páginas no Gruki apenas foi possível, por limitações de tempo, de colocar a funcionar correctamente no ”browser” Mozilla Firefox de versão igual ou superior a 2.0.0.14. Todas as restantes funcionalidades deverão funcionar em vários ”browsers” nomeadamente o Microsoft Internet Explorer 7.0, o Opera 9.5 e o Safari 3.1.1. O Gruki será desenvolvido para ser correctamente apresentado na resolução 1024 por 768 pixels As ”cookies” do ”browser” deverão estar activas.

Interfaces de *software*

O Gruki foi desenvolvido sobre a ”framework” Ruby on Rails o que obriga à existência quer do interpretador Ruby 1.8.6 quer da versão 2.0.2 do Ruby on Rails e respectivas dependências (pacotes ActiveRecord, ActiveSupport, ActiveResource, ActionMailer e ActionPack). O Ruby on Rails irá correr sobre o servidor Apache sendo o ”deployment” feito recorrendo à aplicação mod_rails.

Interfaces de comunicação

A aplicação comunicará entre o browser e o servidor através do protocolo HTTP, sendo usados os protocolos TCP/IP para comunicação e transporte na rede.

2.2.2 Requisitos de usabilidade

O Gruki será desenhado de forma a que a sua utilização seja simples, elevando assim os padrões de usabilidade. Quanto maiores os níveis de usabilidade maior o número de potenciais utilizadores e melhor a experiência de utilização. Esta necessidade é crucial pois num meio como o da ”web” as aplicaçoes mais usáveis e que proporcionam melhor experiência de utilização são aquelas que adiquirem um maior número de utilizadores. As questões da simplicidade de uso do Gruki e da sua usabilidade são tão importantes que nos voltaremos a debruçar sobre elas no capítulo seguinte.

2.2.3 Requisitos de confiabilidade

O Gruki deverá ser confiável e deverá manter a coerência do sistema de permissões limitando o acesso de utilizadores a funcionalidades de acordo com a política de cada espaço.

2.2.4 Requisitos de disponibilidade

A disponibilidade de uma aplicação ”web” depende sobretudo das tecnologias em que está assente, nomeadamente a base de dados e o servidor ”web”. Assim, a disponibilidade do Gruki dependerá essencialmente das aplicações sobre as quais será executado. Contudo, o Gruki deverá apresentar tolerância a erros provocados pelo utilizador (como erros no preenchimento de formulários, por exemplo), permitindo a recuperação desses erros e não compromentendo a integridade da aplicação ou dos dados.

2.2.5 Requisitos de portabilidade

O Gruki é altamente portável, facto que deriva de ser construido sobre uma ”framework” que corre sobre uma linguagem ”open source” que é compatível com vários sistemas operativos. Além da portabilidade da aplicação do ponto de vista do servidor, a aplicação deverá correr correctamente em vários ”browsers”.

2.2.6 Requisitos de escalabilidade

Sendo uma aplicação com um elevado potencial de crescimento, o Gruki deverá conseguir suportar o escalamento para um maior número de utilizadores.

Tal como com os requisitos funcionais, os requisitos não funcionais devem igualmente ser numerados e classificados pela sua . Na tabela seguinte vemos o resultado dessa numeração e prioritização:

Tabela 2.2: Tabela de especificação e ordenação de requisitos não funcionais.

Número	Requisito	Prioridade
1	Requisitos de interfaces externas	Obrigatória
2	Requisitos de usabilidade	Alta
3	Requisitos de confiabilidade	Alta
4	Requisitos de disponibilidade	Média
5	Requisitos de portabilidade	Baixa
6	Requisitos de escalabilidade	Baixa

Capítulo 3

Estado da Arte/Revisão Tecnológica

3.1 O conceito de *wiki*

A principal ideia por trás do conceito de *wiki* é a de uma página *web* que pode ser editada por qualquer pessoa que o pretenda.As implicações desta ideia são, desde logo, imensas. Não será difícil perceber como esta ideia coloca o papel de criação/edição de conteúdos não num conjunto de pessoas que depois as serve mas, virtualmente, em toda a gente.

O exemplo mais conhecido de uma *wiki* é o caso da Wikipedia uma enciclopédia *online* aberta à colaboração de todos. Ao contrário de uma enciclopédia estática como por exemplo as que temos impressas em casa, em que conjuntos de especialistas produzem os conteúdos dessa mesma enciclopédia, obrigando a que esta tenha de ser actualizada periodicamente, a Wikipedia está aberta à edição de todos, especialistas ou não, levando uma maior partilha de conhecimento. Na Wikipedia, todo o conhecimento é relevante e nada me impede de ir à Wikipedia criar uma página sobre o Gruki. Ao imaginarmos o crescimento do Gruki, a comunidade iria verificar a acuidade das informações por mim fornecidas e adicionar mais informações, gerando-se dinâmica e partilha de informação.

Este tipo de colaboratividade em "comunidade"é uma das grandes bases do fenómeno conhecido por Web2.0, termo cunhado para se referir a toda uma vaga de modernas aplicações *web* que oferecem espaço para que criemos conteúdos, ao invés dos conteúdos. Para além da já referida Wikipedia, são exemplos muito populares deste tipo de aplicações Web2.0 o Flickr (para alojamento de fotografias) ou o YouTube (para alojamento de videos).

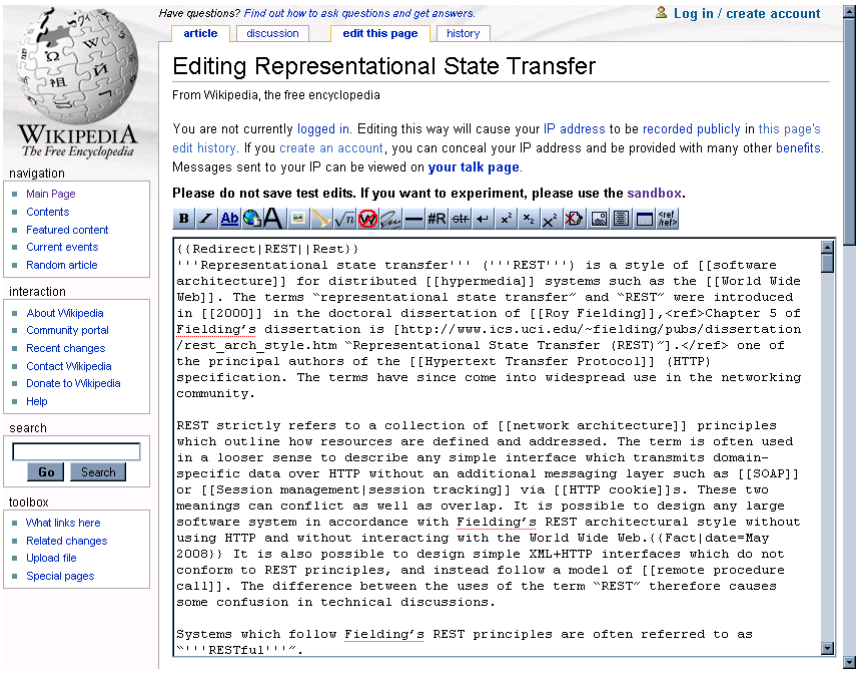

Figura 3.1: Exemplo de uma página da Wikipedia em modo de edição

Ward Cunningham, criador da primeira *wiki*, a WikiWikiWeb, desenvolveu-a para partilhar informações sobre *design patterns* entre os membros do Portland Pattern Repository. Para a sua construção seguiu, inconscientemente, onze princípios fundamentais que mais tarde veio a definir. Uma *wiki* deve então ser:

- Aberta - Uma página que seja considerada incompleta ou mal organizada deverá poder ser editada por qualquer leitor, da forma que este desejar
- Incremental - As páginas devem poder referenciar outras, incluindo páginas que ainda não tenham sindo escritas
- Orgânica - A estrutura e o texto devem ser abertos à edição e evolução
- Mundana - Um pequeno número de convenções textuais (irregulares) deverão oferecer acesso à maior parte da formatação útil
- Universal - Os mecanismos de edição e organização são os mesmos que os de escrita de forma a que qualquer autor seja ao mesmo tempo editor e organizador
- Transparente - O conteúdo formatado (e impresso) deve sugerir qual a sintaxe necessária para o reproduzir
- Unificada - Os nomes das páginas deverão pertencer todas ao mesmo conjunto para que não sejam precisas informações contextuais para a sua compreensão

- Precisa - As páginas deverão ter um título suficientemente preciso para evitar a maior parte da concorrência de nomes, tipicamente formando sintagmas nominais
- Tolerante - Comportamentos não interpretáveis (mesmo que indesejados) são preferíveis a mensagens de erro
- Observável - A actividade de cada página deverá poder ser observada e revista por qualquer visitante da página
- Convergente - A duplicação deverá ser desencorajada ou removida através da citação de conteúdo similar ou relacionado

Estes princípios são os utilizados originalmente por Ward Cunningham mas não são estanques. Questões como a editabilidade por todos os leitores de uma página *wiki* podem provocar actos de vandalismo, bastante comuns, por exemplo, na Wikipedia. Outra questão relevante é a necessidade do conhecimento de uma sintaxe adequada para a edição de uma página. Este conhecimento pode por vezes limitar a universalidade de acesso a uma *wiki* pois o tipo de sintaxe, apesar de bastante comum e facilmente entendível para os membros do meio informático, não é igualmente intuitiva para os restantes potenciais utilizadores. Podemos ainda referir que o princípio da "Unificação"de uma *wiki* pode também ser esquecido em deterimento da criação de páginas contextuais.

Estas três questões são aqui referenciadas pois justificam funcionalidades basilares e distintivas do Gruki: a existência de "espaços", grupos de páginas, que partilham permissões, ajudam a evitar situações de potencial vandalismo (através do bloqueio da edição de páginas) mas também permitem a criação de páginas de nomes iguais mas em contextos diferentes. Por outro lado o Gruki usa um editor WYSIWYG (What You See Is What You Get) cuja utilização não obriga a um conhecimento da sintaxe da *wiki*, permitindo maior acessibilidade e simplicidade na utilização.

Normalmente, além da possíbilidade de edição as *wikis* costumam permitir a visualização de um histórico de todas as versões de uma página o que se torna útil para ajudar à recuperação de situações de vandalismo bem como para verficação das alterações efectuadas numa página ou para recuperação de uma versão antiga tornando-a na versão actual.

O uso das *wikis* tem vindo a crescer de forma exponencial. As suas utilizações são cada vez mais variadas e um número cada vez maior de diferentes motores de *wiki* aumentam a sua flexibilidade em termos funcionais. As *wikis* são muito usadas como base para a gestão e partilha de conhecimento bem como para suporte ao processo de desenvolvimento de software, desde os requistos à fase de entrega, facilitando largamente o processo de escrita de documentação de forma colaborativa. Há ainda uma extensa aplicação das *wikis* como CMS (Content Management System) devido à sua simplicidade na criação de conteúdos.

Em termos futuros as *wikis* são olhadas com bons olhos devido ao seu imenso potencial. A sua utilização como ferramenta de ensino, trazendo para a sala de aula todas as vantagens da *wiki*, como a sua abertura e facilidade de edição a par com os serviços de comunidade que esta oferece, é um bom exemplo de uma futura utilização que seria de todo benéfica. Sendo um conceito recente, a *wiki* tem ainda um longo caminho a percorrer até ser globalmente aprovado.

Do ponto de vista tecnológico as *wikis* são ferramentas não muito complicadas. Sendo composta por páginas web, o acesso a uma *wiki* pode fazer-se apenas através de um browser. Isto é mais uma das vantagens que tornam as *wikis* plataformas tão acessíveis pois um browser *web* é uma aplicação bastante comum existente em quase todos os sistemas operativos.

Arquitecturalmente as *wikis* usam bases de dados ou ficheiros para guardar os seus conteúdos, cabendo ao servidor a função de transformar esses conteúdos escritos com sintaxe *wiki* para HTML interpretável pelo *browser*. Esta questão não é obrigatória (apesar de ser a mais comum), havendo algumas *wikis* que permitem a inserção de código HTML directamente nos conteúdos.

Um conceito importante neste domínio é o de *wikifarm*, um serviço de alojamento de *wikis*, permitindo várias instâncias de *wikis* num mesmo servidor. Em termos tecnológicos isto facilita o trabalho de manutenção e actualização do motor *wiki*, permitindo uma concentração absoluta nos conteúdos.

Existem *wikis* implementadas em quase todas as linguagens de programação, sendo que as linguagens de scripting são as mais comuns.[1]

3.2 A Usabilidade na *web*

Define-se usabilidade como uma medida da facilidade com que se consegue utilizar uma determinada interface. Segundo Jakob Nielsen, um especialista em usabilidade, esta é definida por 5 componentes essenciais[Nie03]:

- Aprendizagem: Quão fácil é para os utilizadores conseguirem executar tarefas básicas da primeira vez que são confrontados com o *design*?
- Eficiência: Uma vez aprendido o *design*, quão rapidamente conseguem executar tarefas?
- Memorização: Quando os utilizadores voltam a usar o *design* após um período de tempo sem o usar, quão facilmente retomam a sua proficiência?
- Erros: Quantos erros os utilizadores cometem, quão severos são, e quão facilmente recuperam deles?
- Satisfação: Quanto prazer se retira da utilização do *design*?

A usabilidade é um termo bastante genérico que se aplica virtualmente a quase tudo aquilo que é passível de ser usado. Assim, vamos nesta secção concentramo-nos na usabilidade para a *web*, ignorando as questões da usabilidade noutras áreas.

Mais uma vez citando Jakob Nielsen, "Na *web*, a usabilidade é uma condição necessária para a sobrevivência. Se um sítio *web* é difícil de usar, as pessoas abandonam-o".[Nie03] Esta verdade quase tautológica resume porque devem as nossas aplicações *web* possuir elevados padrões de usabilidade: evitar o abandono dos utilizadores e fomentar o seu regresso. Num universo como o

[1] Uma lista exaustiva de motores de *wiki* pode ser encontrado em http://c2.com/cgi/wiki?WikiEngines

da web, toda a concorrência está a poucos cliques de distância, o que leva um utilizador frustrado com a sua experiência de utilização a encaminhar-se para qualquer outro sítio[**?**].

No seu livro "Don't Make me Think"[Kru06], Steve Krug, um conhecido consultor de usabilidade, refere a sua lei principal da usabilidade como sendo aquela que dá título ao livro: don't make me think (não me façam pensar). Esta ideia baseia-se no facto de que, numa página *web*, as coisas devem ser evidentes. Se para uma aplicação *desktop* o utilizador está mais aberto à leitura de um manual de utilização, ninguém quer ter de ler um manual de utilização para uma aplicação *web*. Mais uma vez, num meio onde a concorrência está à distância de um clique, uma página deve evitar ao máximo frustrar o utilizador.

Boas práticas de usabilidade levam a que os componentes de uma página sejam evidentes. Ao olhar para uma página, o utilizador deverá ser capaz de identificar (sem ter de pensar) as áreas de navegação e de conteúdos, os sítios onde pode clicar, a página em que se encontra, etc. Apesar de por vezes os utilizadores até serem tenazes o suficiente para usar uma aplicação mesmo que esta seja frustrante e difícil, se esta for evidente e auto-explicativa, toda a aplicação parecerá melhor e proporcionará uma consideravelmente mais agradável user experience.

De entre as várias maneiras de estudar a usabilidade de uma aplicação, sem dúvida que a mais eficaz é o teste da aplicação por utilizadores reais. Não é difícil de conceber que para uma equipa de desenvolvimento, envolvida desde o início num projecto, toda a aplicação parecerá extremamente fácil e usável uma vez que para eles não haverá segredos quanto ao seu funcionamento. Assim, dando a utilizadores reais da aplicação que não tiveram contacto anterior com ela uma oportunidade de a usarem, podemos adquirir valiosas informações sobre as dificuldades encontradas por um utilizador ao utilizar a aplicação.

Existe hoje um paradigma que coloca a responsabilidade da criação de conteúdos para a *web* nos utilizadores. Neste paradigma os wikis têm um papel primordial devido às suas possibilidades de edição. Assim, é essencial para os *wikis* que estes possuam elevados valores de usabilidade. Num artigo publicado em 2005 [DPV05] foram apresentados os resultados de um estudo efectuado sobre a usabilidade dos *wikis*. Neste estudo os principais problemas de usabilidade encontrados prendem-se com a gestão de hiperligações e a orientação dos conteúdos a criar para o formato hipertexto. Apesar disso, as conclusões obtidas referem que apesar de todos os problemas encontrados os *wikis* são ferramentas usáveis, tendo permitido nos testes efectuados a um grupo de crianças, por exemplo, desenvolver de forma colaborativa uma história na *web*.

3.3 Soluções similares existentes

Nesta secção descrevemos algumas soluções similares ao Gruki. Uma comparação deste tipo de *wiki* com qualquer *wiki* não é justa: a lista de motores *wiki* é extensa e seria exagerado referir-me a todos eles. Igualmente, pretende-se com o Gruki abordar outros paradigmas de funcionalidade como a criação de conteúdos potencialmente mais estáticos como seria, por exemplo, uma página pessoal, não esquecendo que a edição WYSIWYG de conteúdos é igualmente relevante.

Assim, escolhi para analisar três aplicações que têm funcionalidades similares ás que se pretendem implementar no Gruki: o Google Sites, o Wikispaces e o Backpack. Iremos ver quais as funcionalidades de cada uma destas soluções e fazer uma análise crítica do seu funcionamento.

3.3.1 Google Sites

O JotSpot[2] foi desenvolvido por uma pequena equipa e começou por ser um serviço de alojamento de *wikis* Actualmente a tecnologia JotSpot foi comprada e adaptada pela Google, tendo agora o nome de Google Sites.

A grande diferença entre o JotSpot e os demais concorrentes era o seu poder para a criação de aplicações dentro das próprias páginas da *wiki*, alargando as fronteiras do conceito. Dentro de uma *wiki* JotSpot, além das várias páginas de conteúdo estático facilmente criáveis numa *wiki*, podia extender-se as funcionalidades da nossa *wiki* adicionando-lhe aplicações que permitiam por exemplo a criação de *blogs*, foruns, galerias de imagens ou calendários. Todos estes conteúdos podiam ser facilmente editados recorrendo ao editor WYSIWYG integrado nas aplicações JotSpot.

Em 2006 a Google comprou o JotSpot e em Fevereiro de 2008 lançou o Google Sites. O Google Sites utiliza o mesmo editor WYSIWYG do JotSpot bem como o mesmo conceito de páginas com aplicações. Neste momento o Google Sites apresenta um número de aplicações para páginas muito menor que o JotSpot, ignorando assim uma das suas mais poderosas funcionalidades. Contudo, os *widgets* adicionáveis às várias páginas no Google Sites permitem adicionar funcionalidades que, em alguns casos, colmatam falhas que a menor quantidade de aplicações para páginas oferece.

A utilização do Google Sites é gratuita e todos os conteúdos são alojados pelo Google. É incentivado o uso de conteúdos produzidos noutras aplicações do Google como o Google Docs ou o Google Spreadsheats através de uma integração nativa do Google Sites com documentos produzidos nestas aplicações.

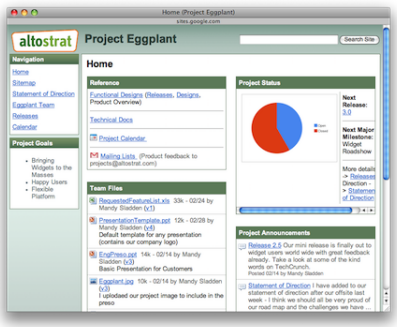

Figura 3.2: Exemplo de uma página criada com o Google Sites

[2]http://www.jot.com/

3.3.2 Wikispaces

Esta aplicação é uma das principais influências do Gruki. O Wikispaces[3] foi desenvolvido pela Tangient[4], uma empresa de desenvolvimento *web* sediada em São Francisco. As funcionalidades do Wikispaces são extensas, sendo de destacar a simplicidade que mantém apesar de todas as suas funcionalidades. Possui um editor WYSIWYG apesar de também permitir a edição manual de código *wiki*. Possui funcionalidades de histórico e de diferenciação entre versões. De forma a facilitar a inclusão de conteúdos possui um gestor de ficheiros para permitir o upload de imagens e outros tipos de ficheiros. Inclui igualmente métodos para fazer backup de um espaço, podendo fazer-se download das páginas em versão HTML. Inclui também notificações de alterações feitas a páginas via email e via RSS bem como todo um conjunto de funcionalidades adicionais cujo volume seria.

O Wikispaces oferece a criação de *wikis* abertas gratuitamente sendo as opções de *wikis* privadas pagas. Oferece cinco tipos diferentes de planos, do gratuito com oferta de 2GB de espaço para a *wiki*, até ao plano que custa 800 USD mensais e que oferece 200GB de espaço.

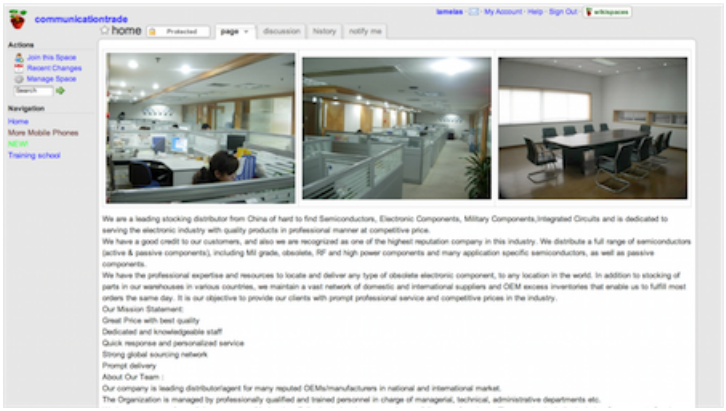

Figura 3.3: Exemplo de uma página do Wikispaces

3.3.3 Backpack

O Backpack[5] é uma aplicação *web* desenvolvida pela 37signals[6] para ser usada como gestor de informação pessoal ou para servir como *intranet* de uma pequena empresa. A 37signals é a empresa por trás da criação do Ruby on Rails pelo que falaremos dela com mais detalhe mais à frente.

No Backpack, grupos de utilizadores partilham um conjunto comum de funcionalidades. A principal funcionalidade do Backpack é a de criar páginas em que podem ser incluídas notas, listas de afazeres, galerias de imagens entre outras coisas. Possui ainda um módulo de calendário que

[3] http://www.wikispaces.com/
[4] http://tangient.com/
[5] http://www.backpackit.com/
[6] http://www.37signals.com/

permite a existência de um calendário partilhado entre os vários utilizadores. Apesar de ser uma aplicação interessante não possui um editor WYSIWYG e obriga ao conhecimento de uma sintaxe própria (ao estilo *wiki*) para aplicar formatações. Uma das mais interessantes funcionalidades do Backpack é possuir um endereço de correio electrónico associado a cada página e que permite que sejam colocados conteúdos numa página através de *email*.

O Backpack é disponibilizado de forma gratuita com um conjunto de funcionalidades reduzido sendo as versões mais completas oferecidas através de um pagamento mensal.

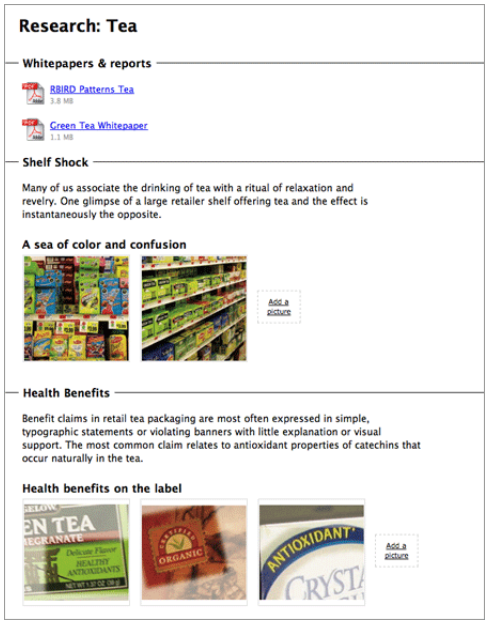

Figura 3.4: Exemplo de uma página criada no Backpack

3.4 Tecnologias utilizadas

Nesta secção descrevemos as tecnologias usadas no desenvolvimento do Gruki. Primeiramente fazemos uma descrição da linguagem de programação Ruby sobre o qual o Ruby on Rails foi desenvolvido. Esta introdução ao Ruby serve para situar a linguagem no contexto das modernas linguagens de programação com o intuito de tentar explicar a sua escolha para o desenvolvimento da *framework* Ruby on Rails em deterimento de outras linguagens largamente usadas para desenvolvimento *web* como o PHP.

Depois fazemos uma descrição do Ruby on Rails, um breve resumo histórico, alguns exemplos de utilização e uma análise crítica das suas vantagens e desvantagens, justificando a sua escolha para o desenvolvimento do Gruki. Refirmos também algumas *frameworks* existentes que se baseiam no mesmo modelo de funcionamento do Ruby on Rails.

Por fim dos servidores *web* utilizados pelo Ruby on Rails, apenas para referência.

3.4.1 Ruby

A linguagem de programação Ruby começou a ser desenvolvida em 1993 pelo japonês Yukihiro Matsumoto ("Matz").

O Ruby é uma linguagem de programação dinâmica reflexiva, o que lhe permite alterar a sua estrutura durante a execução, e orientada a objectos, um paradigma de programação em que os dados são tratados como objectos que possuem propriedades, sendo a aplicação baseada na interacção entre objectos. Estas características derivam grandemente das duas maiores influências do Ruby, o Smalltalk e o Perl. Num artigo publicado em 2000[**?**], Matz, diz que os programas escritos em Ruby parecem, por vezes, "versões reordenadas e simplificadas de programas Perl", afirmando que removeu a maioria das "armadilhas"do Perl, não negando que algumas novas poderão ter aparecido.

No mesmo artigo, Matz refere os três princípios que o guiaram na criação do Ruby, referindo que considera uma linguagem de programação como uma interface com o utilizador e que, como tal, deviam ser concebidas tendo por base os mesmos princípios, que são:

- **Concisão:** Quero que os computadore sejam meus servos e não meus mestres. Assim, quero poder dar-lhes ordens rapidamente. Um bom servo executa uma grande quantidade de trabalho com uma simples ordem.
- **Consistência:** Como com o tratamento uniforme de objectos, como referido antes, um pequeno conjunto de regras define toda a linguagem Ruby. O Ruby é uma linguagem relativamente simples, mas não demasiado simples. Tentei seguir o princípio da "menor surpresa". O Ruby não é demasiado distinto o que leva a que um programador com conhecimentos básicos de programação possa aprende-lo de forma rápida.
- **Flexibilidade:** Como as linguagens pretendem exprimir linhas de pensamento, uma linguagem não deverá restringir o pensamento humano mas ajudá-lo. O Ruby consiste num pequeno e imutável núcleo de sintaxe e um conjunto arbitrário e extensível de bibliotecas. Como a maior parte das funcionalidades é adicionada por bibliotecas, pode-se tratar classes e objectos definidos pelo utilizador da mesma forma que se tratam os existentes de série.

O Ruby herda do Smalltalk a componente de programação orientada a objectos pura, em que tudo é tratado com um objecto, incluindo as primitivas como os caracteres e os inteiros. Não é permitido em Ruby herança múltipla, apenas herança simples. De forma a resolver o problema das confusões motivadas pela herança múltipla, mas não limitando a sua funcionalidade, o Ruby permite o uso de *mixin's* classes não instanciáveis que adicionam funcionalidades ás classes que as incluem. O Ruby permite ainda **"singleton-methods"**, métodos definidos e pertencentes a uma instância de uma classe.

Tal como todas as modernas linguagens de programação o Ruby inclui excepções para recuperação de erros e *garbage collecting*

A tipificação de dados em Ruby é feita de forma dinâmica, sendo o tipo de uma variável apenas verificado quando esta é interpretada. Isto facilita o trabalho de programação pois permite,

como no caso do Ruby, que uma variável não tenha que ser declarada antes de ser instanciada. Há contudo desvantagens desta aproximação, como é o caso da menor velocidade de execução. As variáveis podem igualmente ser usadas segundo a filosofia do *duck typing* uma filosofia de tipificação dinâmica em que se uma variável "anda como um pato e grasna como um pato então deve ser um pato". Tomando o exemplo literalmente, podemos imaginar que se dois objectos implementam o método "grasnar", é irrelevante a classe a que pertencem pois se ambos "grasnam"então são ambos patos. Esta característica só aumenta ainda mais a facilidade da programação em Ruby.

A sintaxe do Ruby é similar à do Perl e à do Python mas como uma versão melhorada de ambas. Ao contrário do Perl, as variáveis não têm de ser marcadas com um sinal de pontuação antes do nome para definir o seu tipo. Em Ruby usa-se sinais de pontuação antes do nome de uma variável mas para referenciar o âmbito da variável. Ao contrário do Python o código não precisa de ser indentado apesar de a mudança de linha ser interpretada como o fim de uma instrução.

Tal como todas as linguagens de programação o Ruby também tem as suas desvantagens. Uma crítica comum prende-se normalmente com o facto de as variáveis não terem de ser declaradas antes de usadas. Isto pode introduzir erros quando, por exemplo, nos enganamos a escrever o nome de uma variável. Ao tentarmos alterar uma variável, por exemplo, e nos enganarmos a escrever o seu nome, uma nova variável aparece, apesar de a variável anterior, que originalmente queríamos alterar, se manter igual. Isto pode dificultar bastante a caça ao erro em projectos grandes. Outras críticas comuns incluem a menor velocidade de execução quando comparada com outras linguagens e o facto de a sua sintaxe, definida pelo criador como "intuitiva", não ser igualmente intuitiva para todos. Refere-se ainda que existem vários problemas de compatibilidade da versão 1.9 do Ruby com programas escritos em versões anteriores.

Sobre a versão 1.9 do Ruby lançada em Dezembro de 2007, é ainda interessante referir que o código Ruby é agora implementado sobre o YARV, uma máquina virtual[7] para o Ruby, que torna a execução de programas muito mais rápida do que nas versões anteriores.[DeN07]

3.4.2 Ruby on Rails

O Ruby on Rails, também vulgarmente referido como Rails, é uma *framework* desenvolvida com o objectivo de tornar o desenvolvimento e manutenção de aplicações *web* mais fácil. Esta *framework* foi extraída da aplicação Basecamp, uma aplicação de gestão de projectos *online* desenvolvida para a 37signals, uma empresa americana de desenvolvimento de aplicações *web* por David Heinemeier Hansson. Em 2004, após ter extraído do Basecamp o Ruby on Rails, tornou a *framework* pública e desde então o Rails tem atraído um número cada vez maior de seguidores.

De forma a conseguir cumprir os objectivos de um desenvolvimento e manutenção de aplicações *web* mais fácil, o Rails usa a linguagem Ruby (ver secção anterior) e dois princípios que são basilares à sua construção: **"convention over configuration"** ("convenção acima de configuração") e **"don't repeat yourself"** ("não te repitas").

[7] Em ciências da computação uma máquina virtual é uma implementação que executa um programa como se este estivesse a correr numa máquina verdadeira.

O princípio da "convenção sobre configuração"pretende diminuir a responsabilidade dos programadores abstraindo um conjunto variado de decisões que estes têm de tomar. Com este princípio consegue diminuir-se drasticamente as configurações necessárias ao desenvolvimento de uma aplicação. O Rails tem em si um conjunto de valores *standard* que tornam os vários aspectos do desenvolvimento de uma aplicação *web* numa tarefa mais fácil. Uma aplicação Rails tem um conjunto de directorias definidas para a colocação de cada componente da aplicação, incluindo além da aplicação propriamente dita, testes, ficheiros de estilos CSS e Javascript. É possível, contudo, fugir a estas convenções mas as vantagens deste afastamento são discutíveis.

O princípio de "não te repitas"diz que nada pertencente à base de conhecimento do sistema deverá ser escrito mais do que uma vez. Este princípio aplica-se ao código, mas também aos requisitos ou à arquitectura da aplicação. As grandes vantagens desta aproximação são, desde logo, evidentes: a maior facilidade nas alterações e a menor quantidade de código, requisitos, funcionalidades, repetidas. Não é difícil conceber que é muito mais fácil e muito menos sujeito a erros alterar, por exemplo, a implementação de uma função do que ter de fazer alterações em vários sítios diferentes, por vezes em ficheiros e até módulos diferentes da aplicação. Os efeitos das falhas provocadas pela repetição podem oscilar desde simples erros a falhas totais do sistema.

No capítulo seguinte iremos ver com mais detalhe a arquitectura de uma aplicação Ruby on Rails, baseada no padrão arquitectural Model-View-Controller. Resumidamente, este padrão tenta separar a interface com o utilizador de toda a lógica de negócio subjacente. Os *models*(modelos) são responsáveis pela manutenção dos dados e de toda a lógica a eles associada; as *views*(vistas) estão apenas responsáveis pela apresentação da interface ao utilizador; os *controllers*(controladores) são os componentes que detectam, recebem e tratam eventos, interagindo com os modelos e seleccionando qual a vista a exibir. De forma a ajudar toda esta interacção o Rails possui vários pacotes[8] que facilitam esta arquitectura como são o **`Active Record`** (orientado para os modelos e a interacção com a base de dados) e o **`Action Pack`**, que se divide entre o **`Action Controller`** (responsável pelos controladores e, portanto, com toda a recepção e gestão de eventos) e o **`Action View`** (que contém todas as funcionalidades para a renderização de templates, sejam eles HTML, XML ou JavaScript). Estes pacotes serão igualmente vistos com maior detalhe no capítulo seguinte.

Sendo uma *framework* para desenvolvimento ágil de aplicações, o Rails oferece vários geradores de código fonte. Estes geradores oferecem suporte para geração de controladores e modelos da nossa aplicação, bem como funcionalidades de *scaffolding*[9] de um modelo, criando ficheiros que oferecem ao utilizador funcionalidades de **CRUD** (`Create-Read-Update-Delete`) sobre um determinado modelo, podendo servir como base para posterior desenvolvimento.

O Rails oferece ainda suporte para o uso de duas bibliotecas JavaScript, a biblioteca Prototype

[8] O RubyGems é um gestor de pacotes para a linguagem de programação Ruby que oferece um formato *standard* para instalação de programas e bibliotecas. Estes pacotes denominam-se por "gems". Em http://www.rubygems.org há mais informações sobre o RubyGems e seus pacotes.

[9] A palavra inglesa *scaffold* significa "andaime"e, assim, funcionalidades de *scaffolding* são funcionalidades de suporte ao desenvolvimento, tal com os andaimes na construção civil.

e a biblioteca script.aculo.us. A biblioteca Prototype é usada essencialmente para o processamento de chamadas AJAX e manipulação do DOM de uma página. Por sua vez a biblitoeca script.aculo.us é usada para animar a interface com o utilizador de forma a tornar a sua experiência de utilização mais agradável. Um exemplo clássico disto é o processamento de uma chamada AJAX. As chamadas AJAX permitem que uma página envie para o servidor um request sem que o endereço actual da página seja alterado e sem que esta seja recebida na sua totalidade. A resposta de uma chamada AJAX pode conter várias coisas como HTML ou código JavaScript que podem alterar a página actual. Como estas chamadas são efectuadas sem que o utilizador se aperceba é igualmente fácil que os seus resultados lhe passem despercebidos. Assim, por um lado a Prototype faz as chamadas AJAX e trata de agir quando recebe uma resposta, a script.aculo.us pode mostrar um efeito de forma a chamar a atenção do utilizador para as alterações efectuadas. As utilizações do AJAX são extensas e a integração da Prototype com o Ruby on Rails facilita imenso a sua utilização. Esse assunto é de âmbito bastante mais alargado e já originou várias publicações, motivo pelo qual não será feita uma descrição mais pormenorizada.

Para facilitar a disponibilização de serviços *web* o Rails tem um suporte bastante alargado para a arquitectura REST (Representational State Transfer - Transferência de Estado Representacional). A arquitectura REST utiliza o protocolo HTTP para transferir informação entre o cliente e um servidor, um pouco como os *browsers web* acedem a páginas. Como o protocolo HTTP não possui estado, toda a informação necessária à transferência de estado está contida pedido. Esta informação de um pedido incluí, por exemplo, quatro verbos - GET, PUT, POST, DEL - que indicam a acção a executar pelo o pedido. A arquitectura REST está hoje bastante divulgada, sendo muito utilizada para permitir acesso às aplicações *web*, possibilitando, através da junção de resultados retornados de várias aplicações, a criação de *mashups*.

Não sendo o Ruby uma linguagem difícil de aprender e de usar, o Rails acaba por sofrer um pouco em termos de aprendizagem devido ao extenso número de convenções. A dificuldade com que se desenvolve uma aplicação simples é muito reduzida mas qualquer aplicação que fuja um pouco do espectro normal do Rails será bastante difícil e envolverá uma necessidade de extensa pesquisa apesar de, posteriormente, o resultado não ser particularmente complicado. Outro dos problemas do Ruby on Rails é a sua estrutura fixa que, apesar de bastante útil, acaba por vezes por se tornar exageradamente complicada.

Assim, o Ruby on Rails é uma *framework* para o desenvolvimento de aplicações *web* que goza de grande popularidade e, digamos até, de um certo factor "cool"dentro do meio, tendo gerado um enorme grupo de seguidores. A sua curva de aprendizagem acaba por limitar um pouco a velocidade de desenvolvimento nas fases iniciais devido ao grande número de convenções que é necessário conhecer.

Sendo uma *framework* em expansão e com uma história relativamente curta, podemos apenas especular quanto ao seu futuro mas, a avaliar pelas capacidades de desenvolvimento ágil permitidas e pela simplicidade presente no Rails, o futuro adivinha-se auspicioso.

3.4.3 MySQL

O MySQL é um sistema de gestão de bases de dados relacionais *open source* Era originalmente desenvolvida por uma pequena empresa Sueca, a MySQL AB, tendo sido comprada pela Sun Microsystems em Fevereiro de 2008. Goza de grande popularidade, contando com mais de 100 milhões de cópias distribuídas por todo o mundo, devendo-se a sua grande popularidade essencialmente ao facto de ser bastante segura e rápida, oferecendo uma alternativa gratuita extremamente viável aos SGBD proprietários como o Oracle ou o Microsoft SQL Server. Existe contudo uma versão Enterprise do MySQL, vocacionada para empresas, com maior foco na assistência técnica por parte dos especialistas da MySQL AB.

O seu estatuto *open-source* permite que existam versões do MySQL para quase todas as plataformas, bem como módulos que permitem a sua utilização em quase todas as linguagens de programação, como o PHP ou o Ruby ou o Python. Sendo *open source* é usado no popular conjunto de aplicações LAMP - Linux, Apache, MySQL e PHP/Python/Perl, que permite o desenvolvimento e instalação de uma aplicação *web* completa apenas baseada em software *open source* e, assim, gratuito.

Hoje em dia muitas aplicações *web* têm suporte MySQL nativo. Isto inclui motores de wiki como por exemplo o MediaWiki (usado pela Wikipedia e restantes produtos da Wikimedia Foundation), CMS's open source como o Joomla! ou o Drupal, bem como o motor de blogs Wordpress.

3.4.4 Servidores *Web*

O conceito de servidor *web* refere-se não só à aplicação que recebe e processa pedidos HTTP mas também, por vezes, ás máquinas cuja funcionalidade é a de correr uma aplicação de servidor *web* mas, salvo referência em contrário, será sempre à aplicação que nos iremos referir. Existem hoje em dia várias servidores *web* capazes de receber e processar pedidos HTTP, sendo que alguns deles podem ser integrados com o Ruby on Rails.

O Rails possui, de série, o Webrick que foi usado para efeitos de desenvolvimento do Gruki. Para efeitos de *deployment* da aplicação foram usados o Apache e o Mongrel. Será feita a seguir uma breve descrição de cada um deles, com algum detalhe.

- Webrick - desenvolvido em Ruby por Masayoshi Takahashi e Yuuzou Gotou com a ajuda de vários utilizadores. É parte integrante do Ruby desde a versão 1.8 e sendo pouco estável mas leve é o ideal para ambientes de desenvolvimento de Rails

- Mongrel - servidor *web* desenvolvido por Zed A. Shaw igualmente em Ruby. Quando instalado como uma RubyGem no sistema, é o servidor predefinido do Rails para execução. O Mongrel usa um sistema de apenas uma *thread* por *request* o que pode levar ao seu sobrecarregamento rapidamente, fechando as conexões com os pedidos que não consegue tratar. De forma a aumentar a performance de aplicações Rails a correr utilizando o Mongrel, usam-se várias instâncias do Mongrel para distribuir o tratamento dos *requests*

- Apache - o servidor HTTP Apache é um dos servidores HTTP mais conhecidos em todo o mundo. É desenvolvido pela Apache Software Foundation mas, sendo *open source* o seu desenvolvimento está aberto a todos. O seu código fonte é escrito em C e inclui interfaces para gerar conteúdo dinâmico em várias linguagens de programação como o Python ou o PHP. Existe uma RubyGem chamada "mod_rails"(também conhecida por Passenger) que permite a utilização de aplicações Rails sobre servidores Apache.

3.4.5 Wedit

O Wedit é um editor de conteúdos *web* desenvolvido por um grupo de alunos da FEUP no âmbito da disciplina de Laboratório de Gestão de Projectos. Nesta disciplina prentendia-se que os aluno tivessem contacto com o mundo empresarial desenvolvendo aplicações reais, para empresas reais, para resolver problemas reais. O Wedit foi então desenvolvido para a PT Inovação e há um interesse na sua utilização uma vez que este foi apenas concebido para gerar conteúdos *web* ficando toda a responsabilidade da lógica subjacente a esses conteúdos noutra aplicação a desenvolver. Assim, o Wedit vai ser usado como o editor de conteúdos do Gruki.

Figura 3.5: Modo de edição de uma página com o Wedit

Como estamos a falar de criação de conteúdos através de um *browser* o Wedit, tal como os a maioria dos editores existentes, foi desenvolvido recorrendo à tecnologia JavaScript. O JavaScript é uma linguagem de *scripting* que é executada num *browser* permitindo a manipulação

do Document-Object Model, uma norma orientada a objectos para representação de documentos HTML.[10]

O JavaScript é hoje uma das linguagens de programação com maior crescimento na sua utilização uma vez que a sua utilização permite o desenvolvimento de aplicações *web* de funcionamento muito similar a aplicações *desktop*. As aplicações *web* representam uma vantagem em relação ás aplicações *desktop* devido à sua interoperabilidade entre *browsers* o que permite a sua utilização em qualquer ambiente, independentemente do *browser* ou sistema operativo.

Apesar da existência de normas, por vezes há diferenças entre a maneira como vários *browsers* interpretam o código Javascript, potenciando uma perda da portabilidade das aplicações *web*. Para eliminar os efeitos dessas diferenças, existem *frameworks* Javascript que abstraem as funcionalidades de manipulação do DOM.

Devido ao elevado número de funcionalidades avançadas presentes, o Wedit utiliza duas dessas *frameworks*: o jQuery[11] e a combinação Prototype+Script.aculo.us.

Entre as funcionalidades do Wedit contam-se a possibilidade de criar/editar:

- Parágrafos
- Títulos (até três níveis)
- Listas numeradas e não numeradas
- Listas de afazeres
- Imagens
- Tabelas

Todos estes elementos podem ser copiados e colados e ver a sua ordem alterada através de *drag and drop* Permite-se ainda a **formatação** dos textos a inserir recorrendo a sublinhados, negritos, itálicos e alteração do alinhamento do corpo de texto.

Com o intuito de fomentar a sua utilização, a inclusão do Wedit numa página *web* é muito simples bastando apenas descomprimir a biblioteca e inserir na página a editar dois comandos seguintes:

```
<script type="text/javascript" src="./wedit/wedit.js"></script>
<script type="text/javascript">
    // <![CDATA[
        Editor([elemento que possui os elementos a editar]);
    // ]]>
</script>
```

[10] Na verdade o DOM pode ser genericamente aplicado a documentos XML, referindo-se que o (X)HTML é um dos seus dialectos.

[11] O jQuery (disponível em http://www.jquery.com) é uma framework Javascript com funcionalidades similares ao Prototype. Uma vez que é apenas utilizada pelo Wedit e não pelas aplicações Rails, apenas se refere superficialmente.

Esta inclusão torna editáveis todos os elementos que estejam contidos dentro do elemento passado por parâmetro ao construtor do editor. Caso sejam elementos que o Wedit reconhece e sabe editar, permite e edição de conteúdos; no caso de ser um elemento que o Wedit não reconhece, apenas permite as funcionalidades de copiar/colar, remoção e alteração de posição.

Refira-se aqui que de forma a testar esta facilidade de integração, antes do desenvolvimento do Gruki, foi desenvolvido um *plugin* para a TikiWiki[12] em que se permite a edição de páginas recorrendo ao Wedit.

[12]Uma *wiki* que possui um enorme conjunto de funcionalidades que a torna quase um CMS. Entre as suas muitas funcionalidades encontram-se, além da *wiki*, blogs, foruns, galerias de imagens, etc. Está disponível em http://www.tikiwiki.org

Capítulo 4

Descrição da Solução

Neste capítulo é detalhada a solução desenvolvida. Abordaremos a arquitectura do Gruki e especificaremos a base de dados que serve de suporte ao Gruki. São também descritos os principais problemas levantados e respectivas soluções encontradas.

4.1 Arquitectura

As aplicações Rails são baseadas numa arquitectura denominada Model-View-Controller (MVC). Esta arquitectura foi originalmente descrita por Trygve Reenskaug em 1979, sendo ainda hoje muito popular: é usada pelo Ruby on Rails, a framework usada para o Gruki, mas também por outras frameworks de desenvolvimento web como o Django (em Python) ou o Cocoon ou o Struts (ambos em Java) bem como várias frameworks para desenvolvimento de aplicações não web como o GTK+ ou o Java Swing. A arquitectura MVC faz uma separação mais clara das responsabilidades das várias camadas da aplicação, permitindo separar a lógica de negócio da interface gráfica e dos dados persistentes.

Conceptualmente a arquitectura MVC é simples, possuindo apenas três componentes. Primeiramente o **Modelo** representa toda a informação específica da aplicação. O modelo acrescenta significância aos dados, tratando igualmente de garantir a integridade dos dados e toda a lógica subjacente. Imaginemos que o nosso modelo iria ser uma representação de uma das páginas do Gruki. O modelo de uma página estaria responsável não só pelo armazenamento dos conteúdos da página mas tambéem como de garantir que todas as informações dessa página seriam válidas como por exemplo se a página em questão estava associada a um espaço ou não. Todas as outras funcionalidades associadas a uma página estariam igualmente contidas no modelo da página. Esta abstracção coloca no modelo toda a responsabilidade em termos de gestão dos dados da aplicação.

A **Vista** é a representação visual de um modelo. Através da vista o utilizador pode ver e manipular o modelo, podendo haver várias representações de um modelo. A página do nosso

exemplo anterior tanto pode ser mostrada como uma página HTML como em qualquer outro formato, dependendo dos dados que sejam guardados pelo modelo.

Por fim o **Controlador** orquestra o comportamento da aplicação. Numa vista o utilizador despoleta eventos que são interpretados pelo controlador. Estes eventos são então tratados de forma a manipularem o modelo e a actualizarem a vista actual ou retornar uma nova, correspondente à vista do resultado do evento produzido.

A adequação desta arquitectura ao desenvolvimento de aplicações web é notória. Nas aplicações web há uma linguagem própria para as vistas que é interpretada pelo *browser* (normalmente HTML) e há uma linguagem própria que é interpretada pelo servidor (seja ela PHP, Ruby, Python, etc.). Só esta separação já é suficientemente forte para que faça sentido a aplicação desta arquitectura. Assim, não é de estranhar que seja usada pela framework Ruby on Rails. O MVC permite uma separação das três camadas lógicas de uma arquitectura: as camadas de interface, lógica de negócio e persistência de dados (ver secção 4.1.1).

Cada aplicação Rails tem uma estrutura definida que facilita a implementação da arquitectura Model-View-Controller. Esta estrutura consiste num conjunto de directórios que é criado originalmente numa aplicação Rails.

Os directórios existentes numa aplicação Rails são:

- **`app`** - Dentro da directoria "app"existem quatro directorias. Uma delas é pouco relevante neste caso e contém dentro os *helpers*[1] da aplicação. As outras três contém, respectivamente, os modelos as vistas e os controladores da aplicação.

- **`config`** - Na directoria "config"definem-se as várias configurações relevantes para a aplicação: os vários ambientes de desenvolvimento, já definidos por convenção ("development", "test"e "production") mas podem ser alterados de forma a satisfazer as nossas necessidades. Estes ambientes (e os demais que podemos configurar) são úteis devido ao facto que em diferentes estados de uma aplicação, são necessários diferentes ambientes de execução. Em modo de desenvolvimento é importante que toda a informação relativa a erros seja registada e mostrada no ecrã mas por exemplo em modo de produção os erros não devem ser mostrados ao utilizador. Na directoria "config"são igualmente configuradas as ligações à base de dados, sendo que estas ligações são diferentes para cada um dos ambientes de execução. Aqui é também configurado o ficheiro de *routing*, responsável pela definição dos endereços URL que a aplicação irá aceitar, e que são introduzidos pelo utilizador no *browser* (descreve-se em 4.3 a maneira como estes endereços são interpretados).

- **`db`** -A directoria "db"possui as informações da base de dados, nomeadamente os scripts das migrações. As migrações são uma útil ferramente do Rails que permitem que a interacção com a base de dados seja feita abstractamente, através de instruções Ruby. Em 4.2 analisa-se com mais detalhe estas migrações e a sua ligação com a base de dados.

[1] Os *helpers* são funções Ruby que são usadas para reduzir ao máximo a lógica de negócio presente nas vistas.

- **`doc`** - Esta directoria contém documentação gerada automaticamente pelo RDoc, uma ferramenta de geração de documentação em formato HTML do estilo JavaDoc, em que os comentários ao código fonte são usados para fornecer informações.
- **`lib`** - Directoria que possui código partilhado por várias áreas da aplicação.
- **`log`** - Localização dos *logfiles* da aplicação, um por cada ambiente definido.
- **`public`** - Todos os conteúdos da aplicação que serão acessíveis através do browser são colocados dentro da directoria "public". Estes conteúdos envolvem todos os ficheiros de JavaScript, as bibliotecas usadas pelo Ruby on Rails (o Prototype e o Script.aculo.us) e os ficheiros CSS que adicionam estilos às vistas HTML. Nesta directoria são guardadas igualmente as imagens usadas pela aplicação (por exemplo nas suas vistas) ou ficheiros que sejam disponbilizados para download. As configurações do Apache da aplicação são também guardadas nesta directoria.
- **`script`** - A directoria "script"contém vários scripts úteis para a geração de código. Estes scripts geram os esqueletos quer dos modelos, quer dos controladores, bem como dos testes associados quer a um quer a outro. Também permitem gerar um sistema de CRUD recorrendo a um modelo já existente, facilitando o desenvolvimento da aplicação através da construção de um simples "andaime"de suporte a funcionalidades básicas.
- **`test`** - Localização dos ficheiros para os testes unitários, funcionais e de integração
- **`tmp`** - Directoria que aloja ficheiros temporários.
- **`vendor`** - Directoria para instalação de plugins

4.1.1 Arquitectura Lógica

Nesta secção é apresentada a estrutura da aplicação. A arquitectura lógica mostra o sistema decomposto em três camadas logicamente separadas, a camada de interface, a camada de lógica de negócio e a camada de acesso a dados. A utilização do Ruby on Rails, devido à sua implementação da arquitectura MVC, já facilita esta separação.

- Na camada de interface ficam localizadas as vistas do Gruki, ficheiros HTML interpretados de forma a apresentarem os resultados pretendidos. Juntamente com as vistas ficam também o Action View, a biblioteca de suporte à geração de vistas.
- Na camada de lógica de negócio ficam os controladores do Gruki, responsáveis por executar as acções pedidas pelo utilizador e de controlar toda a lógica de negócio. Os controladores são subclasses da classe Action Controller de quem herdam métodos e propriedades mas possuem igualmente os métodos que implementam a lógica de negócio do Gruki..
- Na camada de acesso a dados ficam localizados os modelos do Gruki que são construidos sobre o ActiveRecord que recorre a uma abstracção sobre a base de dados.

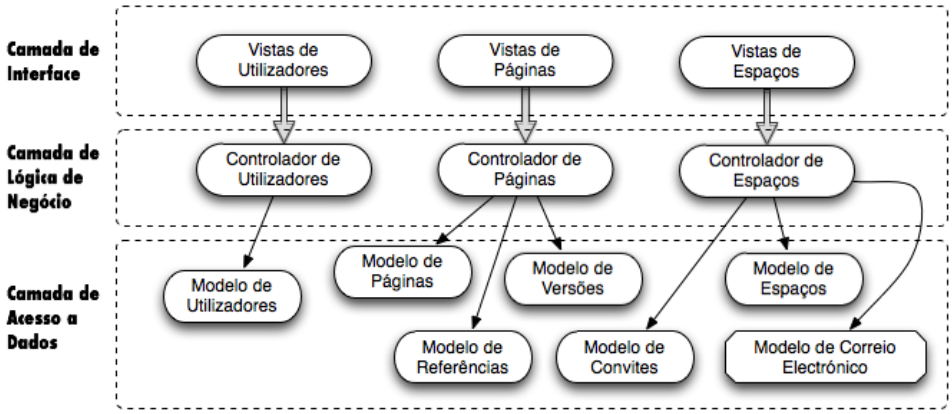

Figura 4.1: Arquitectura lógica do Gruki

Na figura observa-se que os eventos gerados na camada de interface são redirecionados para o controlador correcto que, após executar a lógica de negócio subjacente à acção pretendida, irá manipular os modelos da aplicação (ou não), actualizando posteriormente a vista do utilizador. A separação da aplicação em várias camadas permite-nos observar que os controladores não partilham todos o acesso sobre os mesmos modelos o que faz com que haja uma separação lógica da aplicação também verticalmente, sendo que as acções do controlador de utilizadores, apenas interagem com as vistas e o modelo de utilizador.

4.1.2 Arquitectura Tecnológica

Descreve-se a seguir a arquitectura tecnológica do Gruki. Esta arquitectura é a arquitectura comum a todas as aplicações desenvolvidas em Ruby on Rails, podendo apenas haver algumas variações.

Como se pode observar pela figura o utilizador acede ao Gruki fazendo pedidos HTTP através da utilização de um *browser web* com acesso ao servidor que está a executar o Gruki. O servidor redireciona esses pedidos para os controladores adequados, fazendo-os executar as acções pedidas, de acordo com o *routing* das aplicações Rails. Mais á frente iremos ver como ocorre este redirecionamento. No nível mais baixo da aplicação temos o acesso à base de dados MySQL que suporta o Gruki.

Entre os pedidos efectuados e o acesso à base de dados encontra-se a aplicação propriamente dita, a correr sobre a *framework* Ruby on Rails que corre sobre a linguagem Ruby. Este servidor pode correr em qualquer plataforma para o qual existam implementações do Ruby, como várias versões do Windows, o OS X e várias distribuições de Linux. A portabilidade obtida com esta possibilidade de execução multi-plataforma é uma das grandes vantagens do desenvolvimento de aplicações *web* em Rails.

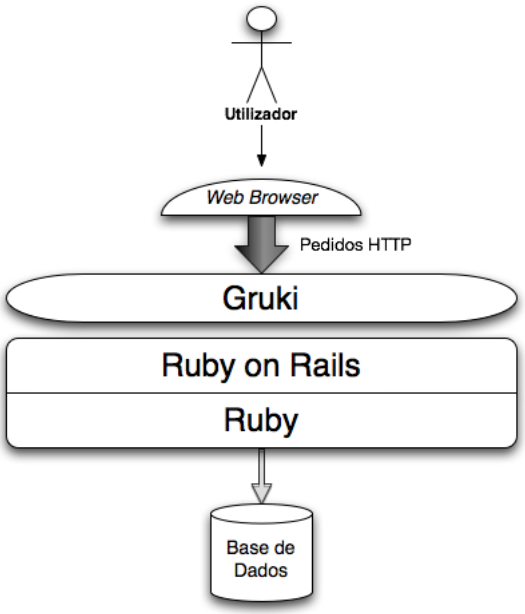

Figura 4.2: Arquitectura tecnológica do Gruki

4.2 Base de dados

Os dados da aplicação são guardados recorrendo a uma base de dados relacional. Por defeito, o Gruki recorre a uma base de dados MySQL para guardar os dados. O Rails facilita toda esta interacção com a base de dados através de uma poderosa abstracção que é conseguida através do sistema de migrações e da biblioteca ActiveRecord. Esta abstracção permite que toda a interacção com a base de dados seja feita sem ter de se recorrer aos mecanismos clássicos de gestão de base de dados.

Numa aplicação Rails é necessário configurar o acesso à base de dados, fornecendo as informações mais comuns necessárias para criar uma conexão à base de dados. Estas informações consistem no endereço do servidor, no nome da base de dados e no nome de utilizador (e respectiva password) que possui acesso à base de dados a usar. Há igualmente necessidade de fornecer uma informação adicional que é a identificação do adaptador de base de dados a usar. Este adaptador consiste numa camada de abstracção que se coloca entre a aplicação Rails e a base de dados. Assim, alterando este adaptador podemos facilmente alterar o tipo de base de dados a utilizar, passando de MySQL para SQLite, Oracle, Postgres ou qualquer outra cujo suporte esteja incluido no Ruby.[2] Após a configuração do acesso à base de dados, está tudo pronto para começar a manipulação da base de dados. Em Rails a base de dados é manipulada ao nível da estrutura através de migrações. Uma migração é um ficheiro de código Ruby que provoca alterações sobre a base de

[2]Sendo o Ruby on Rails uma framework open-source, existe actualmente suporte para um grande número de bases de dados desenvolvido pela comunidade de utilizadores. Uma lista de adaptadores existentes pode ser vista aqui: http://wiki.rubyonrails.org/rails/pages/DatabaseDrivers

dados. As migrações permitem que sejam feitas alterações à base de dados de forma incremental. Assim, podemos ter a nossa primeira migração que irá criar uma tabela e podemos ter a segunda migração em que efectuamos alterações sobre essa tabela, sejam essas alterações inserção de dados ou alteração à estrutura da tabela. As migrações possuem um método que permite que se faça o rollback da migração, podendo voltar assim a uma versão anterior da base de dados.

Apresenta-se a seguir um exemplo de uma migração, nomeadamente a migração que cria a primeira versão da tabela que guarda as páginas do Gruki:

```
class CreatePages < ActiveRecord::Migration
  def self.up
    create_table :pages do |t|
       t.column :name, :string
       t.column :content, :text
       t.column :updated_on, :timestamp
       t.column :created_on, :timestamp
       t.column :lock_version, :integer, :default => 0
    end
   end

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

Este sistema de migrações é altamente benéfico por vários motivos. Desde logo permite que um mesmo script de manipulação de base de dados seja executado em qualquer base de dados, bastando para isso alterar o adaptador a usar. Isto permite, por exemplo, que usemos uma base de dados SQLite para o processo de desenvolvimento e uma base de dados MySQL no servidor de deployment sem precisarmos de alterar nada no processo de criação da base de dados. Por outro lado, por muito bem que a base de dados seja especificada inicialmente, esta sofrerá alterações ao longo do tempo e se, seguirmos o modelo de Agile Development, estas alterações serão ainda mais comuns. Desta formas as migrações permitem agilizar o processo de gestão da base de dados porque para além da migração possuem um método para voltar atrás, eliminando a migração. As migrações são referenciadas por um número de ordem, que é a chave da ordem das migrações. A utilização desse número garante que nenhuma migração será executada antes de outra, garantido a sua ordem lógica e consequente coerência da aplicação ao nível da base de dados.

De forma a facilitar o acesso ás bases de dados o Rails possui além das migrações a biblioteca Active Record. Esta biblioteca contém um enorme conjunto de funcionalidades comuns a toda a lógica relacionada com os Modelos da arquitectura MVC.

Assim, os Modelos a implementar são na verdade subclasses da classe ActiveRecord, herdando todas as suas capacidades. Estas funcionalidades contém métodos que permitem que todo

o processo de acesso à base de dados seja feito através de métodos Ruby dentro de um objecto. Exemplificando, uma página do Gruki será uma instância do modelo Página, permitindo que toda a lógica de CRUD de uma página seja feita sem ser necessário recorrer ao SQL. Igualmente o ActiveRecord facilita questões como a da validação de dados, seja com a obrigatoriedade da existência de um campo ou da necessidade de um valor estar contido num certo intervalo, impedindo a gravação de um objecto na base de dados quando este não passar por todos os métodos de validação.

De forma esquemática, os dados de suporte ao Gruki estão estruturados da seguinte forma:

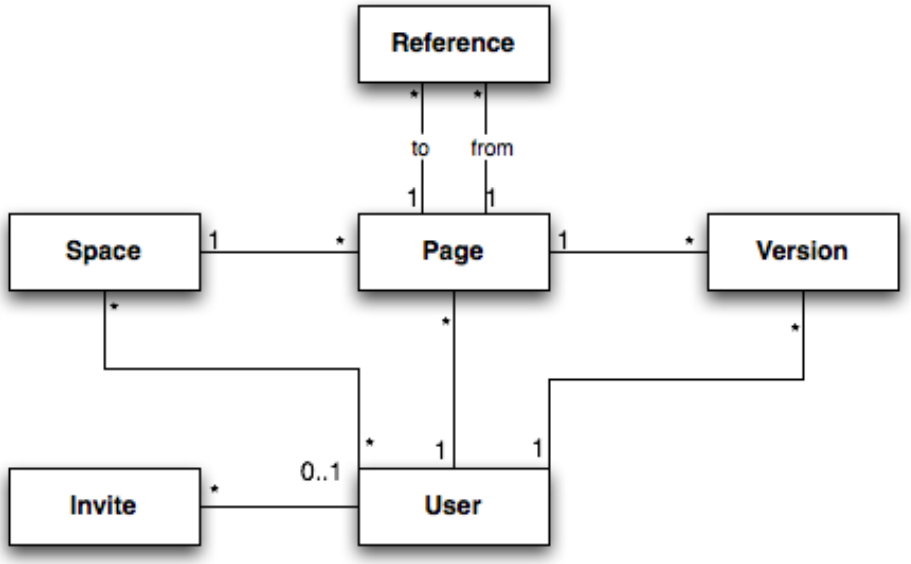

Figura 4.3: Modelo de dados do Gruki

Deste diagrama podemos inferir as relações entre os vários tipos de objectos com que lidamos no Gruki. Podemos assim observar que um espaço possui várias páginas que possuem várias versões. Cada versão só pode pertencer a uma página e cada página a um espaço. Um utilizador está associado a várias páginas e a várias versões de páginas bem como a um ou vários espaços. Por fim, um convite para que um utilizador se junte a um espaço está sempre associado a um espaço mas por vezes, no caso de ser um convite a um utilizador já registado no sistema, esse convite está igualmente associado a um utilizador. A relação potencialmente menos óbvia é a de uma página e as suas referências: uma página pode referenciar outras e pode igualmente ser referenciada por outras.

Este esquema de dados foi posteriormente mapeado numa base de dados relacional, sendo atribuidas a cada classe os seus atributos como colunas de uma tabela.

Estas tabelas foram automaticamente criadas pelas migrações do Rails, tendo sido apenas fornecidos alguns campos. Algumas colunas têm nome especiais que levam o Rails a agir de forma diferente para com essas colunas.

A tabela "pages"guarda as informações sobre as páginas:

Tabela 4.1: Especificação da tabela pages

Campo	Tipo	Descrição
id	integer	Chave primária da tabela, identficador numérico único de uma página
name	string	Cadeia de caracteres que contem o nome de uma página
content	text	Conteúdo da página
space_id	integer	Valor número do identificador do espaço a que a página pertence
author_id	integer	Identificador numérico do autor da última versão da página
visits	integer	Número de visitas recebidas por uma página desde a sua criação
updated_on	datetime	Data e hora do último update da página, este campo é preenchido automaticamente pelo Rails quando actualiza uma linha na base de dados
created_on	datetime	Data e hora da criação da página, preenchido quando uma linha da base de dados é criada

A tabela "spaces"guarda as informações sobre os espaços do Gruki:

Tabela 4.2: Especificação da tabela spaces

Campo	Tipo	Descrição
id	integer	Chave primária da tabela, identficador numérico único de um espaço
name	string	Nome do espaço.
admin_id	integer	Identificador numérico do utilizador que possuir privilégios de administração neste espaço
permissions	integer	Valor inteiro que guarda o valor das permissões do espaço

Na tabela "users"são guardados os dados dos utilizadores registados:

Tabela 4.3: Especificação da tabela users

Campo	Tipo	Descrição
id	integer	Chave primária da tabela, identficador numérico único de um utilizador
email	string	Cadeia de caracteres com o endereço de correio electrónico do utilizador
username	string	Nome do utilizador registado, é com este nome que é efectuado o *login*
hashed_password	string	Valor da palavra passe encriptada
salt	string	*Salt* de suporte à geração da chave encriptada
created_on	datetime	Data e hora do registo de um utilizador. Este campo é automaticamente preenchido pelo Rails ao criar o registo.

Para que utilizadores sejam associados a espaços e espaços a utilizadores, o Rails reconhece uma tabela com o nome "spaces_users"como tabela de ligação entre ambos. Esta tabela não possui uma chave primária.

Tabela 4.4: Especificação da tabela spaces_users

Campo	Tipo	Descrição
space_id	integer	Identificador do espaço
user_id	integer	Identificador do utilizador

A tabela que guarda as referências chama-se "references"e guarda pares de identificadores de páginas.

Tabela 4.5: Especificação da tabela references

Campo	Tipo	Descrição
from	integer	Identificador da página que referencia outra
to	integer	Identificador da página que é referenciada

As versões de uma página são guardadas na tabela "versions".

Tabela 4.6: Especificação da tabela versions

Campo	Tipo	Descrição
id	integer	Chave primária da tabela, identficador numérico único de uma versão
content	integer	Conteúdo da versão
created_on	datetime	Data em que a versão foi criada, preenchida automaticamente pelo Rails
page_id	integer	Identificador da página a que esta versão pertence.
author_id	integer	Identificador do autor da versão

A última tabela chama-se "invites"e guarda os convites para ingressão num espaço.

Tabela 4.7: Especificação da tabela invites

Campo	Tipo	Descrição
id	integer	Chave primária da tabela, identficador numérico único de um convite
user_id	integer	Quando um utilizador já registado é convidado, este campo guarda o seu identificador.
user_email	string	Quando um utilizador não registado é convidado, este campo guarda o valor do seu endereço de correio electrónico
space_id	integer	Identificador do espaço para o qual o convite diz respeito
invite_hash	string	Cadeia de caracteres com a chave única de cada convite
created_on	datetime	Data e hora da criação do convite

4.3 Detalhes de implementação

Nesta secção irá ser descrita a aplicação de um ponto de vista mais profundo. Iremos detalhar quais os modelos e quais os controladores existentes no Gruki, fazendo uma descrição das funcionalidades de cada um deles. Não estão aqui incluidas as vistas uma vez que em Rails, por cada método de um controlador, existe normalmente uma vista associada.

Os controladores são a base funcional de uma aplicação Rails pois é neles que todas a lógica de negócio se realiza. Ao receber um pedido de um *browser*, o Rails faz o encaminhamento desse pedido para o controlador adequado que por sua vez irá executar o método pedido e posteriormente gerar a vista correspondente.

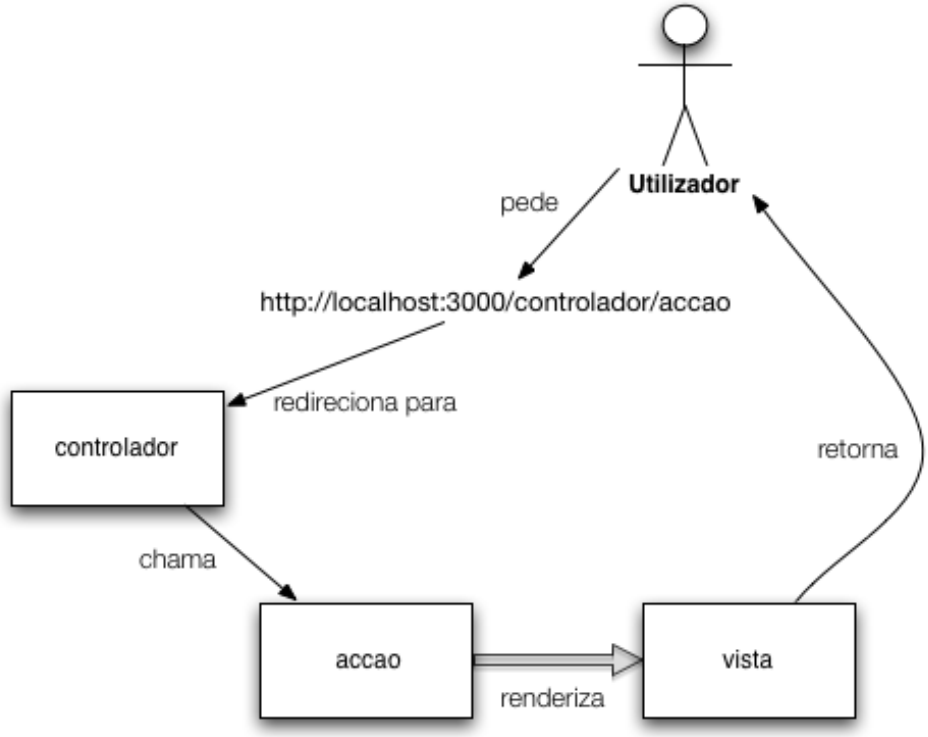

Figura 4.4: Esquema de processamento de um pedido em aplicações Rails

Antes de chamar o controlador pedido pelo *browser* o Rails irá chamar o controlador da aplicação. Este controlador é **sempre** chamado antes de qualquer outro controlador a não ser que seja dado ordem em contrário. No Gruki este controlador é usado para fazer a autorização de um utilizador, verificando antes de executar a acção se esta é permitida ao utilizador. O Gruki reconhece endereços com o esquema de série do Rails:

```
http://[servidor:porta]/[nome do controlador]/[nome da acção]
/[paramêtro]
```

Além destes o Rails reconhece também outros paramêtros passados no endereço. Por exemplo, o endereço de acesso a uma página no Gruki seria:

```
http://[servidor:porta]/page/show/[nome da página]?space=
[nome do espaço]
```

De forma a que as páginas do Gruki apresentem uma estrutura comum, foi definida uma vista que, tal como o controlador da aplicação, é chamada antes da vista pedida. Este *layout* define a estrutura à volta dos conteúdos de cada página, incluindo um formulário para autenticação do utilizador e vários butões com funcionalidades como o acesso ao profile de um utilizador ou a troca do espaço em que se está a trabalhar.

4.3.1 Controladores

O Gruki possui três controladores que executam todas as funcionalidades implementadas. Segue-se uma descrição de cada um deles, bem como das acções por eles executadas e alguns detalhes sobre as vistas de cada um.

4.3.1.1 O Controlador de Páginas

Este controlador é responsável pelas acções relacionadas com as páginas do Gruki.

Ao ser chamado o método "show", será carregada e mostrada a página pedida. Além de mostrados os conteúdos da página, são mostrados butões que permitem acesso às funcionalidades de uma página como a edição, a visualização das páginas que a referem, o histórico de alterações dessa página e a remoção dessa página.

Numa versão mais evoluída do Gruki, só seriam mostradas as acções executáveis pelo utilizador mas, por questões de limitação de tempo do projecto, todas as acções são mostradas. Isto não quer dizer, contudo, que todas as acções sejam executáveis: quando um utilizador tentar aceder a uma acção que não lhe é permitida será redirecionado para uma página indicando que não tem permissões para executar aquela acção. Este princípio é aplicado para todas as acções não permitidas.

O método "edit"é chamado para edição da página. Este método carrega a página como se se tratasse do método *show* mas depois irá carregar o Wedit que vai percorrer os conteúdos da página tornando-os editáveis. O Wedit utiliza a tecnologia AJAX para enviar para o servidor um pedido de forma a gravar as alterações que o utilizador vá efectuando. Estes pedidos são enviados para o método "save"do controlador e são enviados periodicamente de 5 em 5 segundos mas o servidor só irá guardar uma nova versão se tiver havido alterações. Após gravar uma nova versão de uma página, o servidor adiciona à lista de versões antigas dessa página uma nova versão com os conteúdos que vão ser substituídos.

A criação de uma nova página consegue-se através do método "new". No caso de não ser fornecido um nome para a página, é apresentado um campo para que o utilizador possa inserir o nome da página a criar. Ao ser criada uma página esta é criada com um conteúdo que o utilizador deve substituir. Após ser criada a página, é feito um redirecionamento para o método de edição, como se de uma página normal se tratasse.

Ao ser apagada uma página, é chamado o método "delete"que destrói a versão actual da página mas também apaga da base de dados todas as versões antigas de uma página. Assim, o processo de apagar uma página é irreversível.

A visualização das páginas que referenciam a actual é conseguida pelo método "references"que vai procurar todas as instâncias da tabela de referências que possuem uma referência para a página actual.

As últimas funcionalidades de uma página estão relacionadas com a visualização do seu histórico. O histórico lista as versões antigas da página com um link que permite a visualização individual de cada uma destas páginas. Quer na listagem de versões, quer na vista de uma versão,

existe um botão que permite tornar a versão actual da página numa dessas versões anteriores, um processo chamado de *rollback*.

4.3.1.2 O Controlador de Espaços

Neste controlador são geridas as acções relacionadas com os espaços. Estas acções incluem os processos de criação de um espaço e posterior edição das suas permissões. Lembrando o princípio de se querer manter o Gruki como uma aplicação simples, a criação de um espaço apenas envolve a escolha de um nome e das permissões do espaço. Posteriormente, a única propriedade de um espaço que é alterável é o valor das permissões.

Dentro das funcionalidades de cada espaço há várias listagens que são potencialmente úteis:

- lista das páginas pertencentes a um espaço
- lista dos utilizadores associados a um espaço
- lista das páginas de um espaço que não são referenciadas por nenhuma outra

Para facilitar a sua visualização no caso de haver um grande número de elementos, estas listagens são paginadas.

Uma das acções essenciais dos espaços é a possibilidade de convidar utilizadores para se juntarem a um espaço. Para que sejam efectuados convites, o utilizador que criou o espaço (o seu administrador) irá inserir uma lista de *emails* separados por vírgulas. Seguidamente será criado um convite na base de dados associado a cada um desses *emails*. De forma a garantir que cada convite é único é gerada uma chave, utilizando o algoritmo MD5, de uma cadeia de caracteres composta pelo tempo actual e a chave primária do espaço para o qual o convite está a ser gerado. Quando já existir no Gruki um utilizador registado com esse email o convite a enviar será diferente. Após ser criado o convite, é enviado um *email* para o utilizador com o endereço a seguir para aceitar o convite: se o utilizador já estiver registado, será automaticamente adicionado ao espaço; se não estiver registado será levado até ao formulário de registo de um novo utilizador e o convite será posteriormente aceite.

4.3.1.3 O Controlador de Utilizadores

A habitual gestão de utilizadres está também presente no Gruki. Assim, o Gruki suporta as funcionalidades normais às aplicações que possuem registo de utilizadores: o registo, visualização do perfil, edição dos dados e remoção de uma conta. Além destas funcionalidades o controlador de Utilizadores gere o processo de *login* e *logout* de um utilizador. Ao nível dos utilizadores é útil a existência de uma listagem dos espaços a que um utilizador pertence para que possa, facilmente, mudar de um para o outro. Refira-se ainda que mais uma vez a simplicidade foi tida em conta e o registo de um utilizador no Gruki requer apenas um conjunto mínimo de dados.

4.3.2 Modelos

No Gruki os modelos são todos bastante simples mas, ainda assim, iremos fazer uma breve descrição de cada um, explicando as suas funcionalidades e validações que executam.

Nas aplicações Rails os modelos são subclasses da classe ActiveRecord e herdam dela um conjunto de métodos para acesso e interacção com a base de dados. O ActiveRecord possui funções que facilitam o estabelecimento das relações entre modelos, funções cujos nomes tornam a sua compreensão imediata. Assim, quando no modelo de Páginas se refere

```
has_many :versions
```

é imediato percebermos que uma página possui várias versões.

O ActiveRecord possui igualmente métodos para facilitar a validação dos dados de um modelo, como por exemplo para obrigar a que um determinado campo possua um valor. Estes métodos de validação apenas são chamados quando se executa o comando que grava um objecto na base de dados, cancelando essa inserção se a validação não for concluida com sucesso.

O modelo de **páginas** possui então métodos que associam uma página a um espaço e a um autor (através do método "belongs_to") e métodos para associar uma página ás suas versões, páginas que a referem e páginas que por ela são referidas (através do método "has_many"). Uma página deve ser validada de forma a ter obrigatoriamente um nome e conteúdo, sendo também efectuada uma validação que obriga a que o nome da página seja único dentro de um espaço. O modelo das **versões** das páginas apenas serve para manter as referências ás páginas originais e aos autores de uma versão. Existe associado ao modelo de páginas um modelo que guarda as **referências** das páginas umas às outras, impedindo a existência de mais de uma referência de uma mesma página para outra. Este modelo possui um método estático que encontra as referências efectuadas numa página, guardando-as de seguida na base de dados.

No Gruki, os **espaços** possuem um conjunto de páginas e de convites que lhes são associados e que são destruidos quando um espaço é apagado. A relação entre os espaços e os utilizadores é uma relação de "muitos para muitos"e é conseguida através do método "has_and_belongs_to_many"que utiliza a tabela "spaces_users"para manter este relacionamento. Em termos de validação, um espaço tem de possuir um nome, um administrador e um valor para as permissões. Este valor de permissões terá de ser um valor do *array* de permissões que é igualmente definido dentro do modelo de espaços. A cada espaço estão associados vários **convites** que, uma vez que são totalmente criados por controladores, não é necessária nenhuma validação.

Os **utilizadores** do Gruki têm de ver validado seu registo, obrigando o modelo dos utilizadores a que exista um endereço de email válido e um nome de utilizador único bem como uma palavra passe. Dentro do modelo de utilizadores é também tratada a lógica de geração de palavras passe encriptadas. Ao ser criado um utilizador, este fornece uma palavra passe que é associada com um *salt*[3], sendo esta cadeia de caracteres posteriormente submetida ao algoritmo SHA1, sendo guardadas na base de dados a chave encriptada e o valor do *salt*. No modelo de utilizadores é

[3]Neste contexto um *salt* é uma cadeia de caracteres aleatória que é utilizada para garantir mais segurança à geração da chave encriptada.

igualmente feito o processo de autenticação que, após ser fornecida a palavra passe e o nome de utilizador a autenticar, é gerada uma chave utilizando a palavra passe fornecida para autenticação e o *salt* guardado na base de dados para o nome de utilizador fornecido. A autenticação é bem sucedida quando a chave encriptada gerada é igual à chave encriptada guardada na base de dados.

O último modelo que será abordado é na verdade um falso modelo segundo a arquitectura MVC. O modelo ***spacer_mailer*** guarda os modelos das mensagens de correio electrónico que são enviadas quando é enviado um convite. Esta classe não é uma subclasse do ActiveRecord mas da classe ActionMailer.

Capítulo 5

Demonstração

Nesta secção irá ser feita uma demonstração do estado actual do Gruki. Esta demonstração irá consistir em alguns *screnshots* de algumas das funcionalidades principais da aplicação de forma a mostrar como estas se efectuam e qual a interface desenvolvida.

Não é do âmbito deste relatório nem deste projecto um estudo aprofundado sobre o *deployment* de aplicações Rails e, assim sendo, essas questões não serão abordadas: assume-se que o Gruki está a ser executado num servidor e que é possível aceder-lhe através de um *browser*.

Figura 5.1: A interface do Gruki e respectivas áreas lógicas

Como se pode observar pela imagem, o Gruki tem um aspecto limpo. Não foi colocado muito esforço no *design* do Gruki uma vez que essa questão ficava fora do âmbito do trabalho. Contudo, foi feito um esforço para que o aspecto do Gruki fosse minimamente bonito, mantendo sempre a simplicidade que era pretendida. Uma vez que o Gruki não é um produto terminado, existem algumas funcionalidades implementadas que não foram integradas na interface.

Existem quatro áreas distintas na interface do Gruki:

- no canto superior direito estão as acções relacionadas com o utilizador, como o formulário de autenticação e o acesso ao registo. Quando um utilizador está autententicado são mostradas as acções de *logout*, de selecção de um espaço e de criação de um novo espaço bem como o acesso ao perfil do utilizador.
- no canto inferior esquerdo estão as acções de uma página, incluem os botões de acesso à edição de uma página, páginas que referem a actual, histórico de versões e remoção de uma página.
- no canto superior esquerdo estão as acções do espaço actual. Estas incluem a ida para uma listagem das páginas de um espaço e acesso à alteração de permissões de um espaço.
- no centro da página está o conteúdo da página, sendo que esta área varia de tamanho consoante os conteúdos. No canto inferior direito da página fica o nome do autor da última versão da página.

Quando um utilizador não autenticado acede ao Gurki, tem no canto superior direito da página a opção de **registar** uma nova conta. Ao registar-se, tem depois acesso às funcionalidades que apenas ficam disponíveis para o utilizador autenticado. O formulário de registo tem o seguinte aspecto:

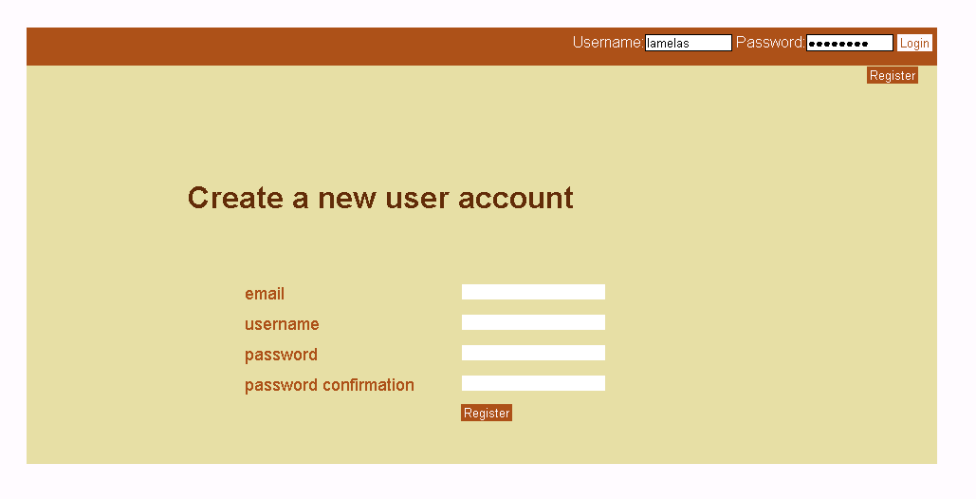

Figura 5.2: Formulário de registo de um utilizador no Gruki

Após o preenchimento deste formulário o utilizador pode autenticar-se no Gruki e começar a criar espaços e páginas, bem como aceder a espaços para os quais tenha sido convidado.

Em modo de **edição de uma página** o Gruki tem o seguinte aspecto:

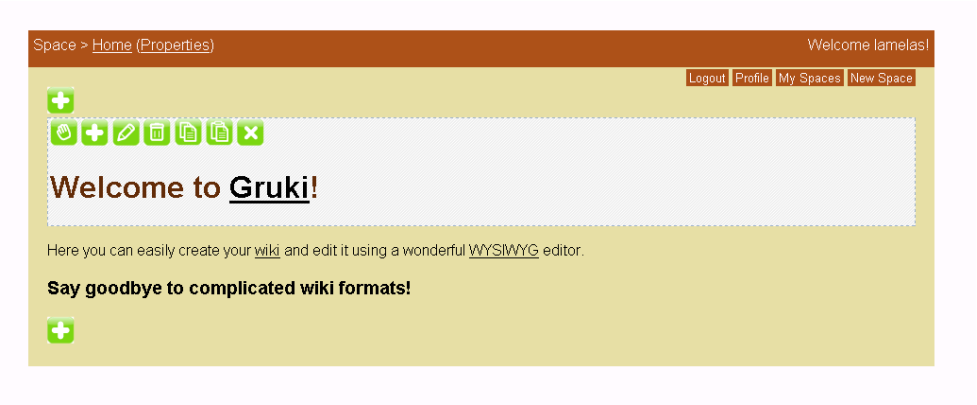

Figura 5.3: Uma página em modo de edição no Gruki

Como se pode observar na imagem, o Wedit faz com que a edição da página seja feita em modo WYSIWYG, o que torna a edição um processo simples e intuitivo. O mdoo de edição de uma página é acedido através do botão "Edit"localizado na parte inferior da página.

Capítulo 6

Conclusões e Trabalho Futuro

Neste último capítulo faz-se uma análise dos resultados obtidos e disserta-se um pouco sobre o cumprimento dos objectivos propostos. Fala-se também também das perspectivas futuras do Gruki e dos seus potenciais desenvolvimentos.

Os *wikis* são hoje uma ferramenta essencial no paradigma de produção colaborativa de conteúdos. Para muitos os usos de um *wiki* estão ainda limitados ao seu uso num estilo como o da Wikipedia, como repositório de conhecimento editável por todos. A verdade é que os usos de um *wiki* são muito mais extensos e variados e estão ainda a ser explorados.

Numa empresa que está fisicamente dividida entre Aveiro, Porto, Lisboa e São Paulo como a PT Inovação e que possui clientes em sítios tão distantes como o Botswana ou Timor, é verdadeiramente importante o papel dos *wikis* como ferramentas para escrita de documentação, *tracking* de tarefas, levantamento de requisitos entre outros usos.

O Gruki foi desenvolvido quase na totalidade tendo em conta os requisitos propostos, sendo que apenas o *dashboard* de actividade não foi implementado a tempo. Será fácil rotular o Gruki como simples mas a verdade é que a utilização da *framework* Ruby on Rails tornou o seu processo mais complicado. O elevado número de convenções existentes no Ruby on Rails tornaram a sua aprendizagem um processo mais moroso e difícil do que inicialmente se esperava. Sendo baseado na arquitectura MVC, a adaptação a este paradigma de programação levou a algum atraso no processo de desenvolvimento.

Após algum tempo de habituação à *framework*, reconheço agora a sua grande versatilidade no desenvolvimento de aplicações *web*. A sua completa separação entre as camadas de interface, lógica de negócio e de tratamento de dados bem como a sua abstração para o uso de sistemas de gestão de bases de dados contam-se entre as inúmeras vantagens encontradas na utilização do Ruby on Rails. Devido ás suas características muito próprias, o Ruby on Rails é uma *framework* em que apenas uma certa habituação e um certo método de trabalho podem levar à produtividade. A sua forte estruturação impede que se seja verdadeiramente produtivo desde o ínicio mas permite igualmente que a sua aprendizagem seja apenas uma questão de tempo e de habituação.

O período de tempo limitado para o desenvolvimento do Gruki levou a que os testes a efectuar sobre a aplicação não tenham sido realizados. Com um maior período de tempo à disposição, teria sido imperativa a realização de testes unitários e funcionais bem como de usabilidade.

Apesar das dificuldades encontradas o Gruki chegou a um bom estado de maturidade, encontrando-se completamente funcional. Contudo, há sempre espaço para a adição de melhoramentos. Normalmente os *wikis* possuem um grande número de funcionalidades tornando-se por vezes exageradamente confusos. Exemplo disso é a TikiWiki que possui além da wiki, fóruns, blogs, galerias de imagens, etc, quer apesar de a tornarem bastante poderosa, permitindo a sua utilização quase como um CMS clássico, tornam-na extremamente confusa para um utilizador que nela se inicia.

O Gruki poderia ser melhorado através da inclusão de funcionalidades que permitissem exportar os dados de um "espaço"para um ficheiro que o utilizador pudesse guardar e também a disponibilização de um *feed* RSS para que mais facilmente se pudessem seguir as alterações efectuadas numa página ou espaço. Uma funcionalidade igualmente útil seria a de permitir ver as diferenças entre duas versões de uma mesma página.

Outros melhoramentos dos quais o Gruki iria beneficiar largamente seriam os melhoramentos ao nível da portabilidade do Wedit. Sendo um editor muito interessante devido à sua natureza WYSIWYG e elevada usabilidade, a sua limitada funcionalidade ao nível dos *browsers*, funcionando apenas totalmente no Mozilla Firefox, faz com que um dos princípais pontos positivos do Gruki se veja limitado por motivos externos.

O lote de funcionalidades extra que podem ser adicionadas ao Gruki é imenso. Há contudo um princípio que me parece relevante manter que é o da simplicidade. Citando John Maeda no seu livro *As Leis da Simplicidade*, "Simplicidade é subtrair o óbvio e adicionar o relevante"[Mae06]. Este princípio de manter apenas aquilo que é relevante deve ser mantido ao longo da evolução futura do Gruki. Um dos grandes problemas ao nível do *software* não usável é o elevado número de funcionalidades que estes contém, tornado-os confusos e difíceis de usar. Algumas coisas não podem ser tornadas nem mantidas simples mas julgo que, dentro do possível, devemos tentar essa simplicidade.

Referências

[BBvBo01] Kent Beck, Mike Beedle, Arie van Bennekum, and othres. Manifesto for agile software development, 2001. Disponível em http://www.agilemanifesto.org/, acedido a 16 de Junho de 2008.

[Bur92] Steve Burbeck. Applications programming in smalltalk-80(tm): How to use model-view-controller? (mvc), 1992. Disponível em http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html, acedido a 20 de Junho de 2008.

[Che06] Nicholas Chen. Convention over configuration, 2006. Disponível em http://softwareengineering.vazexqi.com/files/pattern.html, acedido a 18 de Junho de 2008.

[Cun02] Ward Cunningham. What is a wiki, 2002. Disponível em http://www.wiki.org/wiki.cgi?WhatIsWiki, acedido em 13 de Junho de 2008.

[Cun08] Ward Cunningham. Wiki design principles, 2008. Disponível em http://c2.com/cgi/wiki?WikiDesignPrinciples, acedido a 19 de Junho de 2008.

[DEG+05] G. Dueck, Anja Ebersbach, Markus Glaser, Richard Heigl, and Andrea Adelung. *Wiki: Web Collaboration*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[DeN07] Rick DeNatale. Ruby 1.9 released, 2007. Disponível em http://www.infoq.com/news/2007/12/ruby-19, acedido a 18 de Junho de 2008.

[DPV05] A. Desilets, S. Paquet, and N.G. Vinson. Are wikis usable? *International Symposium on Wikis: Proceedings of the 2005 international symposium on Wikis*, 16(18):3–15, 2005.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[Hun07] Matthew Huntbacho. What's wrong with ruby?, 2007. Disponível em http://www.bitwisemag.com/2/What-s-Wrong-With-Ruby, acedido a 18 de Junho de 2008.

[Hun08] Andrew Hunt. Don't repeat yourself, 2008. Disponível em http://c2.com/cgi/wiki?DontRepeatYourself, acedido a 18 de Junho de 2008.

[Ino08] PT Inovação. Relatório e contas 07, 2008. Disponível em http://www.ptinovacao.pt/pdf/ptinrel2007.PDF, acedido a 23 de Junho de 2008.

REFERÊNCIAS

[KM05] Andrew Koenig and Barbara E. Moo. Templates and duck typing, 2005. Disponível em http://www.ddj.com/cpp/184401971, acedido a 18 de Junho de 2008.

[Kru06] Steve Krug. *Don't Make Me Think - A Common Sense Approach to Web Usability*. New Riders, second edition edition, 2006.

[Mae06] J. Maeda. *The laws of simplicity*. MIT Press, Cambridge, Mass.; London, Eng, 2006.

[Mat00] Yukihiro Matsumoto. The ruby programming language, 2000. Disponível em http://www.informit.com/articles/article.aspx?p=18225, acedido a 17 de Junho de 2008.

[Nie03] Jakob Nielsen. Usability 101: Introduction to usability, disponível em http://www.useit.com/alertbox/20030825.html, 2003. Disponível em http://www.useit.com/alertbox/20030825.html, acedido a 23 de Junho de 2008.

[Pri08] Pjotr Prins. Ruby: Productive programming language, 2008. Disponível em http://www.linuxjournal.com/article/5915, acedido a 18 de Junho de 2008.

[TH07] Dave Thomas and David Heinemeier Hansson. Agile web development with rails, 2007.