Universidade de São Paulo
Instituto de Matemática e Estatística

Trabalho de Formatura Supervisionado

SYSCOMA:
A System for Competence Management

João Marco Maciel da Silva

Supervisor Prof. Valdemar W. Setzer

São Paulo, novembro de 2014

Agradecimentos

Agradeço primeiramente a minha esposa, Simone, pelo apoio moral, incentivador, por todas as horas de paciência, por todas as refeições que pude ajudar pouco a preparar. Agradeço a meu pai pelas orientações de vida. Agradeço a minha mãe pelo apoio financeiro, sem o qual não teria concluído este texto nem o projeto. Agradeço finalmente ao Professor Valdemar W. Setzer que, mesmo a par das minhas dificuldades de tempo e mesmo eu não sendo um bom orientando, não desistiu de apoiar meu projeto, de me orientar e de me fornecer material.

Resumo

da Silva, J. M. M. **SYSCOMA: A System for Competence management**. Monografia (Bacharelado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2014.

No contexto de grandes empresas é comum o surgimento da necessidade de se encontrar um funcionário conhecedor de um determinado assunto ou especialista no mesmo. O *modus operandi* de tais empresas, que não possuem um *software* específico que poderia auxiliá-las, é a abertura de seleção interna ou, até mesmo, a contratação de um novo funcionário exclusivamente para a execução de uma única tarefa, gastando tempo e recursos. O resultado deste projeto é um sistema de código aberto capaz de auxiliá-las nesta busca. Com ele uma empresa pode acelerar o processo de seleção, ajudando na formação de equipes e preenchimento de cargos, bem como na difusão de conhecimento, descobrindo-se quem sabe o quê, quem pode dar treinamento e quem precisa de treinamento.

Palavras-chave: matriz de competência, gerenciamento de competência, conhecimento, competência, gerenciamento, difusão de conhecimentos.

Abstract

da Silva, J. M. M. **SYSCOMA: A System for Competence management**. Monografia (Bacharelado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2014.

For big companies the need of finding an employee who knows something about or is a specialist in some area is quite common. The *modus operandi* of this kind of company that doesn't have a specific software to help in this search is to use common knowledge (including vitae) about its employees or when hiring a new employee, spending time and resources, mainly if the company has hundreds of employees.
The result of this project is a FLOSS (Free, Libre and Open Source Systems) to speed up this kind of process by using competence matrices, a special way of representing knowledge areas and abilities, and searching employees in a competence database. This project can help in selection processes, formation of teams, filling up vacant positions and can help spreading knowledge by allowing finding who knows what, who can transfer knowledge and who needs training.

Keywords: competence matrix, competence management, knowledge, competence, management, .

Índice

Lista de Abreviaturas vi

Lista de Figuras vii

Lista de Tabelas viii

1 Introdução 1
- 1.1 Motivação . . . 1
- 1.2 Considerações Preliminares . . . 1
- 1.3 Objetivos . . . 2
- 1.4 Contribuições . . . 2
- 1.5 Organização do Trabalho . . . 2

2 Conceitos 3
- 2.1 Resumo dos conceitos sobre matrizes de competência . . . 3
  - 2.1.1 Exemplo . . . 3

3 Trabalhos Anteriores 5
- 3.1 Projetos similares . . . 5
  - 3.1.1 PRODESP & PROMON . . . 5
  - 3.1.2 SiGeCo . . . 6
- 3.2 Literatura . . . 7

4 Desenvolvimento 9
- 4.1 Planejamento . . . 9
- 4.2 Ruby x Java . . . 9
- 4.3 Arquitetura e projeto . . . 10
  - 4.3.1 Aplicação das definições . . . 10
  - 4.3.2 Modelo conceitual de dados . . . 10
  - 4.3.3 Estilo arquitetural Rails . . . 11
- 4.4 Testes . . . 15
  - 4.4.1 Fixtures e FactoryGirl . . . 15
  - 4.4.2 Verificação de cobertura de testes . . . 16
- 4.5 Internacionalização . . . 16
- 4.6 Uso da biblioteca Devise . . . 18
- 4.7 Mais sobre Ruby on Rails . . . 18

4.8 Código 18
4.9 Resultado 18

5 Desenvolvimento Futuro **28**
5.1 O que precisa ser feito 28
5.2 O que é desejável que seja feito 28
5.3 O que pode ser feito 29

6 Conclusões **30**
6.1 Considerações finais 30

Referências Bibliográficas **31**

Lista de Abreviaturas

SYSCOMA	A System for Competence Management
AJAX	*Asynchronous* Javascript *and* XML (Javascript Assíncrono e XML)
BDD	*Behaviour-Driven Development* (Desenvolvimento Guiado por Comportamento)
CCSL	Centro de Competência em *Software* Livre
CM	*Competence Matrix* (Matriz de Competência)
ERB	*Embedded* Ruby (Ruby Embutido)
ERP	*Enterprise Resource Planning* (Sistema Integrado de Gestão Empresarial)
FLOSS	*Free, Libre and Open Source System*
HRM	*Human Resources Manager* (Gerenciador de Recursos Humanos)
HTML	*HyperText Markup Language* (Linguagem de Marcação de Hipertexto)
HTTP	*HyperText Transfer Protocol* (Protocolo de Transferência de Hipertexto)
JSON	JavaScript *Object Notation* (Notação de Objeto JavaScript)
JVM	*Java Virtual Machine* (Máquina Virtual Java)
REST	*Representational State Transfer* (Transferência de Estado Representativo)
RVM	*Ruby Virtual Machine* (Máquina Virtual do Ruby)
SQL	*Structured Query Language* (Linguagem Estruturada de Consulta)
TDD	*Test-Driven Development* (Desenvolvimento Dirigido a Testes)
UML	*Unified Modeling Language* (Linguagem de Modelagem Unificada)
URI	*Uniforme Resource Identifier* (Identificador Uniforme de Recursos)
XML	*eXtensible Markup Language* (Linguagem de Marcação Extensível)
YAML	YAML *Ain't Markup Language* (YAML Não é Linguagem de Marcação)

Lista de Figuras

4.3.1 Modelo conceitual de dados . . . 11
4.3.2 Diagrama do padrão arquitetural model-view-controller . . . 12
4.3.3 Diagrama do padrão arquitetural Model 2 . . . 13
4.4.1 Tela da análise de cobertura de testes . . . 16
4.4.2 Tela da análise de cobertura de testes de uma classe . . . 17
4.9.1 Tela de entrada . . . 19
4.9.2 Tela de nova sessão . . . 19
4.9.3 Página inicial de um usuário administrador . . . 19
4.9.4 Página inicial de um usuário comum . . . 20
4.9.5 Página inicial de um usuário gerente . . . 20
4.9.6 Tela de criação e edição de matrizes de competência . . . 20
4.9.7 Tela de exibição de definição de uma matriz . . . 21
4.9.8 Tela de edição de graus de competência das habilidades . . . 21
4.9.9 Tela com a lista das pessoas que preencheram um certo grau de competência . . . 22
4.9.10 Tela de edição de grupo . . . 22
4.9.11 Tela com o resultado da busca por funcionários . . . 23
4.9.12 Tela de escolha da matriz que deseja preencher . . . 23
4.9.13 Tela de escolha da área de conhecimento . . . 23
4.9.14 Tela de escolha dos graus de competência das habilidades . . . 24
4.9.15 Tela de perfil de um usuário . . . 24
4.9.16 Tela de exibição de matriz de competência preenchida . . . 25
4.9.17 Tela de nova busca de pessoas . . . 25
4.9.18 Tela de busca de pessoas, escolha do grau de competência . . . 25
4.9.19 Tela de busca de pessoas . . . 26
4.9.20 Tela de busca de pessoas com buscas armazenadas . . . 26
4.9.21 Tela com o resultado da busca de pessoas . . . 27

Lista de Tabelas

2.1.1 Exemplo de matriz de competência 4

3.1.1 Diferenças entre o SiGeCo e o SYSCOMA 6

4.3.1 Relação entre as classes do projeto e os conceitos utilizados 10

Capítulo 1

Introdução

SYSCOMA é um projeto *open-source* para o gerenciamento de competências. Ele é inspirado em Setzer [1999], onde é definido o que são e como são usadas matrizes de competência e onde se encontram a maioria das definições do capítulo 2.

1.1 Motivação

Em grandes empresas, com quadro de centenas de funcionários espalhados por diversos setores, prédios, até mesmo estados e países, é comum o surgimento da necessidade de se encontrar um funcionário conhecedor de determinado assunto ou especialista no mesmo. O *modus operandi* de tais empresas que não possuem um *software* específico que poderia auxiliá-las nessa tarefa é a abertura de seleção interna ou, até mesmo, a contratação de um novo funcionário exclusivamente para a execução de uma única tarefa. Na ausência desse *software*, as empresas usam em geral a análise de currículos e conhecimento pessoal. O sistema aqui proposto não visa substituir totalmente o julgamento humano, mas auxiliá-lo restringindo o número de pessoas que devem ser examinadas com mais detalhe.
Exemplos de uso desse tipo de projeto são encontrados em Setzer [2000], onde são exibidos os casos da PROMON (Cia. de Processamento de Dados do Município de São Paulo) e da PRODESP (Cia. de Processamento de Dados do Estado de São Paulo). Posto isto, verifica-se que o uso de um modo mais eficiente de encontrar pessoas que detenham certas competências poderia trazer grandes benefícios à empresa e, inclusive, aos funcionários. Nesse último caso, o sistema permite, por exemplo, localizar colegas que possam ajudar no esclarecimento de alguma dúvida.
Existem soluções para isso no mercado, mas a grande maioria é de código fechado, e as duas de código aberto não são muito acessíveis, como demonstrado no capítulo 3.

1.2 Considerações Preliminares

Esse sistema, após a conclusão do curso de MAC0499, *Trabalho de Formatura Supervisionado*, será aberto ao público sob a licença GNU AGPL v3. Assim, será possível o uso, a modificação e o desenvolvimento por qualquer pessoa, desde que respeitando a licença especificada.
É importante salientar que existe uma outra solução de código aberto para o mesmo problema, mas que não é baseada no conceito de matrizes de competência, desenvolvida pela Câmara Federal. Existe também um *software* mais antigo e descontinuado chamado SiGeCo, que também foi desenvolvido na disciplina de MAC0499 sob supervisão do Prof. Setzer, e que usa matrizes de competência, mas não funciona. O sistema aqui descrito não se utilizou de nenhuma parte desse projeto anterior, inclusive no desenho das telas.
A necessidade de um novo sistema gerenciador de competências é devida ao projeto antigo estar defasado, pois ele usa Maven2, que está na versão 3 e usa Java 6, que está na versão 8. Os outros

arcabouços (*frameworks*) e bibliotecas todos estão desatualizados e isso pode ser um problema até de segurança. Além disso não é mais possível uma instalação SiGeCo sem perder horas com a adaptação e atualização da ferramenta pois ele não está em estado operacional.
Este projeto pode ser interpretado como uma especialização de projeto HRM (*Human Resources Manager*, Gerenciador de Recursos Humanos) e uma atividade futura poderia ser integrá-lo com alguma ferramenta aberta existente, como o OrangeHRM. O projeto pode ser visto também como um adicional a sistemas ERP (*Enterprise Resource Planning*, Sistema integrado de gestão empresarial) e é possível pensar em uma futura integração com alguma ferramenta desse tipo. Também pode ser usado como um caso mais geral de aplicações de sistemas para contratações de funcionários, bastando separar de alguma forma o banco de dados dos funcionários atuais do banco de dados dos candidatos a vagas.

1.3 Objetivos

O objetivo deste projeto é criar um sistema baseado na Internet, simples de ser usado, para o gerenciamento de funcionários ou equipes a partir de competências, usando o conceito de matrizes de competência. O sistema resultante é de fácil manutenção e adaptação e é extensível, para que no futuro novas funcionalidades possam ser adicionadas.

1.4 Contribuições

As principais contribuições deste trabalho são as seguintes:

- Um sistema que pode ser empregado para selecionar pessoas por um critério quantitativo de competências;
- Auxiliar no preenchimento de cargos;
- Verificar quem precisa de treinamento dentro de uma empresa;
- Auxiliar na busca de quem tem competência em algo, para troca de conhecimentos e experiências;
- Seleção de pessoal capacitado a dar treinamento em algo dentro de uma empresa.

1.5 Organização do Trabalho

Este trabalho está dividido em 7 capítulos. O capítulo 2 trata dos conceitos usados na idealização do projeto para a consecução do objetivo descrito neste capítulo. A maior parte dos conceitos é oriundo de Setzer [1999].
O capítulo 3 descreve outros trabalhos de código aberto similares a este, mais especificamente o sistema da Câmara Federal e o sistema anterior de gerenciamento de competências SiGeCo. Nesse capítulo são descritas as principais diferenças entre eles e este projeto. Esse capítulo também faz uma resenha da literatura sobre o tema, descreve os principais tópicos que influenciaram a interpretação e a implementação de competências e matrizes de competências neste projeto.
O capítulo 4 trata do desenvolvimento em si, incluindo o planejamento, a arquitetura, as dificuldades encontradas, os testes feitos e o código que foi programado.
O capítulo 5 trata de extensões futuras, do que é desejável que seja feito e do que pode ser feito com este projeto em termos de desenvolvimento.
O último capítulo, 6, contém as conclusões do trabalho e últimas considerações a serem feitas.

Capítulo 2

Conceitos

Este projeto é baseado em Setzer [1999], onde é definido o que são e como funcionam matrizes de competência e de onde são tiradas a maior parte das definições a seguir. Nesse texto são apresentadas uma definição formal de dado e caracterizações de informação, conhecimento e competência.

2.1 Resumo dos conceitos sobre matrizes de competência

Setzer [1999]: **Dado** é "uma seqüência de símbolos quantificados ou quantificáveis", como um texto ou uma foto. Para o autor, "quantificável" significa que, se algo é quantificado, a partir dessa quantificação pode-se reproduzir o objeto original sem perda de detalhe. Uma foto é quantificável, mas uma árvore não é.

Setzer [1999]: **Informação** "é uma abstração informal (isto é, não pode ser formalizada por meio de uma teoria lógica ou matemática), que está na mente de alguém, representando algo significativo para essa pessoa. Note-se que isto não é uma definição, é uma caracterização, porque 'algo', 'significativo' e 'alguém' não estão bem definidos; assumo aqui um entendimento intuitivo (ingênuo) desses termos".

Setzer [1999]: **Conhecimento** é uma "abstração interior, pessoal, de alguma coisa que foi experimentada por alguém."

e Silva e Michaelis [1998]: "**Habilidade** sf (lat *habilitate*) 1 Qualidade de hábil. 2 Capacidade, inteligência. 3 Aptidão, engenho. 4 Destreza. 5 Astúcia, manha."

Área de conhecimento é o conjunto de conhecimentos que podem ser agrupados por compartilharem as mesmas habilidades. Esta é uma caracterização pessoal extraída da interpretação do texto em Setzer [1999].

Setzer [1999]: **Competência** é uma "capacidade de executar uma tarefa no 'mundo real'" e é caracterizada "como a confluência de uma habilidade em uma área de conhecimento, com a respectiva representação matricial".

Setzer [1999]: **Matriz de Competência** é o agrupamento em forma de matriz das áreas de conhecimento (representadas nas linhas) que possuem as mesmas habilidades (colunas). Em cada célula da matriz é inserido um grau de competência, indicando uma quantificação da competência que uma pessoa detém em uma certa área de conhecimento para uma certa habilidade.

2.1.1 Exemplo

Tendo essas caracterizações em mente, é possível dar um exemplo do que seria uma matriz de competência. Um exemplo simples e bem característico seria uma matriz de competência de línguas estrangeiras; esses dados são comumente vistos em currículos e cadastros de empresas. A tabela 2.1.1 é um exemplo de uma matriz de competência de línguas.

		Habilidades		
		Leitura	*Escrita*	*Conversação*
Áreas de Conhecimento	*Inglês*	Fluente	Fluente	Fluente
	Francês	Intermediário	Iniciante	Intermediário
	Alemão	Iniciante	Nulo	Nulo
	Italiano	Intermediário	Intermediário	Avançado

Tabela 2.1.1: *Exemplo de matriz de competência*

Capítulo 3

Trabalhos Anteriores

Este capítulo é dividido em duas partes. A primeira trata de sistemas de *software* livre para o gerenciamento de competências, assim como suas diferenças, vantagens e desvantagens sobre o sistema aqui descrito. A segunda parte trata da literatura sobre o tema.

3.1 Projetos similares

No período de pesquisa sobre sistemas semelhantes existentes foram encontrados diversos com os mesmos objetivos, mas apenas dois eram livres; os outros eram não apenas de código fechado, mas também pagos.
Nenhuma das aplicações de código fechado que são distribuídos na Internet dizem usar matrizes de competência. Não foi possível testá-los por não serem grátis.
Os dois de código aberto são o Banco de Talentos, da Câmara dos Deputados, e o SiGeCo, de outro trabalho de formatura supervisionado pelo Prof. Setzer.
O Banco de Talentos, apesar de ter uma licença GPL (*General Public License*, Licença Pública Geral, todo *software* que participa do programa *Software* Público Brasileiro segue essa licença) não tem o código disponível. É preciso ser registrado como um usuário e possuir uma senha para se ter acesso ao mesmo. Vasculhando o fórum do projeto foi encontrado um arquivo *.war* que foi possível rodar no Tomcat 8 (no manual de instalação diz que funciona a partir da versão 4), porém não há carga de dados. Seguindo o manual de instalação, ele pede para rodar um *script* em SQL, só que não há tal *script*; certamente deve estar junto com o código. Assim, não foi possível testar esse sistema.

3.1.1 PRODESP & PROMON

O únicos trabalhos fechados aos quais houve acesso a alguma informação foram os produzidos pelo Professor Valdemar W. Setzer, supervisor do SYSCOMA, para a PRODESP e para a PROMON. As informações às quais houve acesso foram:

- O modelo de dados. Ele se assemelha bastante ao modelo de dados do SYSCOMA;
- O planejamento de menus. Isso inspirou algumas funcionalidades a mais no SYSCOMA;
- Exemplos de matrizes de competências. Como faltou a pesquisa de campo para criação de novas matrizes, esses exemplos serviram para criar matrizes iniciais para uso do sistema.
- Diretrizes de implementação. Elas ajudaram na questão de requisitos do sistema;
- Telas do sistema. Os exemplos de telas inspiraram particularmente as telas de preenchimento das matrizes de competência.

Os textos foram particularmente importantes por pertencerem a uma implementação de um sistema similar feito pelo próprio autor de Setzer [1999].

3.1.2 SiGeCo

SiGeCo é um sistema similar ao produzido neste projeto. Ele também foi um Trabalho de Conclusão da mesma disciplina (MAC0499) por quatro alunos, supervisionados pelo mesmo professor deste projeto, inspirado pelo mesmo artigo Setzer [1999] e também baseado na Internet.
As diferenças entre os dois projetos são exibidas na tabela 3.1.1 e descritas abaixo.

Diferença	**SiGeCo**	**SYSCOMA**
Funciona	não	sim
Linguagem	Java	Ruby
Arcabouço	algum do Maven2	padrão do Rails
Gerenciador de Dependências	Maven2	Rubygems
Servidor de aplicação	qualquer que suporte JavaEE	Usando máquina virtual Ruby qualquer um que aceite Rails; usando uma máquina virtual JRuby qualquer que funcione Java. Por padrão Passenger
Banco de Dados	Por meio do padrão de projeto DAO, não configurável, usa obrigatoriamente o gerenciador MySQL	Por meio do padrão de projeto ActiveRecord, configurável, usando por padrão como gerenciador SQLite (Hipp [2014])
Internacionalização	Por arquivo de texto, pós interpretado, gerenciado pela própria aplicação	Por arquivo de texto, pós interpretado, gerenciado pelo Rails
Testes	Unitários	Unitários e de sistema
Controle de versão	SVN	GIT
Métricas de código	Checkstyle	Não usa
Análise de cobertura de testes	Não usa	100%
Deploy	manual	automatizado

Tabela 3.1.1: *Diferenças entre o SiGeCo e o SYSCOMA*

Claramente, a diferença mais importante, além da implementação totalmente nova, é que o SYSCOMA funciona. A segunda diferença está na instalação do ambiente de desenvolvimento e de produção. No caso do SYSCOMA basta seguir as instruções que estão no arquivo *README.rdoc*, que está na pasta raiz do projeto, se se desejar usar a configuração padrão, usando RVM (máquina virtual), SQLite (banco de dados) e Passenger (ferramenta para automatizar a colocação em produção) e funciona em Linux pelo menos (e talvez funcione em outros sistemas baseados em Unix). Com um pouco mais de trabalho funcionará em Windows, mas a instalação nesse Sistema Operacional não é coberta pelo projeto. No caso do SiGeCo isso é bem mais complicado, pois o Maven desatualizado torna mais difícil a instalação, alguns pacotes usados não são mais mantidos, e alguns pacotes deixaram de existir nos repositórios do Maven. Assim, seria necessário procurar na Internet e baixar manualmente a dependência (ver adiante). Maven é uma ferramenta de automatização da construção de projetos e inclui diversos arcabouços e configurações do projeto e gerencia dependências. De acordo com Weilkiens e Oestereich [2010] dependência é a relação em UML (*Unified Modeling Language*) que mostra que um elemento, ou um conjunto de elementos, requer um outro modelo de elementos para sua especificação ou implementação; um gerenciador de dependências seria um gerenciador de pacotes onde os pacotes são bibliotecas e arcabouços. Além disso não há

instruções claras para instalação. O SiGeCo tem a vantagem de usar Java, o que significa que é compatível com quase qualquer sistema operacional; outra vantagem é que servidores de aplicação para Java são mais comumente achados em provedores de serviço de hospedagem de mais baixo custo, o que aumenta bastante o número de opções para colocar o projeto em produção. Como há a necessidade de ter instalado o Maven3 por conta de outro projeto que o autor trabalha, isso complicou bastante a instalação pois não é possível ter os dois instalados ao mesmo tempo. Assim, nos testes do SiGeCo, foi necessário executar um Maven compilado e armazenado em alguma pasta do sistema diferente da padrão; isso causou muitos conflitos no sistema.
Outra diferença é o uso de um arcabouço do Maven para iniciar o projeto no SiGeCo. O Maven dá uma quantidade enorme de opções, o que resulta numa variedade grande de arquiteturas. No SYSCOMA é utilizado o arcabouço padrão do Rails, o que significa que se o desenvolvedor já trabalhou em algum projeto com Rails vai entender rapidamente a arquitetura deste projeto.
O SiGeCo usa como gerenciador de dependências o Maven2 e a configuração dele no arquivo *pom.xml*. O SYSCOMA usa como gerenciador de dependências o Rubygems. Não há vantagens ou desvantagens reais entre os dois, exceto pelo fato de o Rubygems estar na versão mais nova e cuidar apenas das dependências, enquanto o Maven também cuida de algumas configurações do sistema.
O servidor de aplicações é também diferente. No caso do SiGeCo pode ser usado qualquer servidor de aplicação Java de acordo com a documentação (mas isso não foi testado pelo autor de SYSCOMA), o que é uma afirmação incorreta, pois existem limitações quanto à versão mínima a ser utilizada. No caso do SYSCOMA qualquer servidor de aplicação recente funciona. No mínimo a versão mais recente do Passenger que é instalada pelo *script* de instalação. Nesse caso o SiGeCo tem uma vantagem pois existem muito mais opções de servidores de aplicação para Java.
O uso da RVM (Ruby Virtual Machine) resolve o problema de ter várias versões do Ruby, do Rails ou do Rubygems na mesma máquina pois, com ele, é possível ter em cada projeto uma versão diferente do interpretador e dos arcabouços.
O uso de banco de dados no SiGeCo é gerenciado pelo arcabouço Hibernate e usa o padrão de projeto DAO. No SYSCOMA, o banco de dados é gerenciado pelo Rails, e usa o padrão de projeto ActiveRecord. No caso do SYSCOMA o uso do banco de dados é mais transparente para o desenvolvedor e a modelagem é mais simples.
Os testes automatizados para o SiGeCo são apenas testes unitários, que verificam separadamente cada componente sem testar as interações entre eles. No SYSCOMA foram incluídos, além de testes de unidade, testes de sistema, que verificam a interação de vários componentes.
Uma vantagem do SiGeCo foi o uso de métricas de código, apesar de não ter sido feita nenhuma análise sobre isso além da verificação de estilo.
Uma vantagem do SYSCOMA é a verificação de cobertura de testes, garantindo o percentual de linhas de código que foram testadas e que não há trechos inúteis no código (isto é, que não são alcançáveis por testes de aceitação ou por um usuário comum), o que ainda não garante ausência de defeitos.

3.2 Literatura

Além de Setzer [1999], que já foi citado e comentado, existem outros artigos que tratam de gerenciamento por competências e matrizes de competência.
Em Brandão *et al.* [2008] os autores usam a definição de Carbone *et al.* [2005] de competências humanas ou profissionais, afirmando que estas são combinações sinergéticas de conhecimentos, habilidades e atitudes, expressas pelo desempenho profissional dentro de determinado contexto organizacional, que agregam valor a pessoas e organizações. Eles citam definições de outros autores. Nesse artigo eles discorrem sobre como medir competências e citam o caso de uso do Banco do Brasil, que emprega desde a década de 1960 medidas para avaliar capacidades de seus funcionários, incluindo gerenciamento por competências.

O próprio Carbone *et al.* [2005], não foi achado, nem sua segunda versão de 2009.
Em Brandão e Guimarães [2001] os autores fazem uma comparação entre duas metodologias de gerenciamento, por competências e por desempenho, discutindo também como usá-las em conjunto. Isso nos leva a pensar numa funcionalidade futura para o SYSCOMA para avaliação de desempenho e talvez uma futura mudança de nome.
Em Brandão e Bahry [2014] os autores discorrem sobre métodos e técnicas de levantamento de competências, os quais não foram incluídos neste trabalho.

Capítulo 4

Desenvolvimento

O desenvolvimento do projeto foi dividido em várias etapas. A primeira foi a pesquisa de trabalhos anteriores, descrita no capítulo 3. A segunda fase foi o planejamento e o grosso das decisões arquiteturais. A terceira foi o desenvolvimento em si junto com os testes. A última parte do desenvolvimento foi a escrita de uma configuração inicial com um usuário administrador inicial.

4.1 Planejamento

O planejamento iniciou-se no começo de março e teve como tarefa inicial criar um cronograma do que seria feito. Decisões inicialmente administrativas geraram previsões de tempo para cada tarefa a ser feita. No fim, por diversos motivos o cronograma inicial foi parcialmente desobedecido, faltando, inclusive, a parte de fazer um estudo para criar matrizes especialmente úteis para o CCSL (Centro de Competência em *Software* Livre).
A primeira decisão importante para o projeto seria escolher uma linguagem e dar motivos para tal escolha. Como se trata de um projeto com muitas entidades foi escolhido como paradigma de programação o da orientação a objetos. Segundo, deveria ser usada uma linguagem orientada a objetos cujos programas fossem fáceis de serem lidos e compreendidos, tivesse uma grande comunidade de desenvolvedores e bons tutoriais, seja amplamente usada para projetos baseados na Internet e, o mais importante, na qual o autor tivesse um mínimo de experiência. Só esses requisitos já limitaram a escolha a 2 linguagens, Ruby e Java. A seguir é elucidado o porquê da decisão de um projeto em Ruby citando-se as vantagens e desvantagens sobre a Java.

4.2 Ruby x Java

Primeiramente as desvantagens de usar Ruby ao invés de Java.

- Tamanho da comunidade. Apesar de a comunidade de Ruby ser relativamente mais ativa que a de Java, a comunidade de Java ganha em número e é, hoje, de acordo com TIOBE [2014] e com PYPL [2014], a linguagem orientada a objetos mais usada no mercado;
- Velocidade. A JVM, principal máquina virtual Java, converte o código fonte em *bytecode*, enquanto Ruby é uma linguagem interpretada na maioria das máquinas virtuais, isto é, interpretada diretamente do código fonte; em geral máquinas virtuais que traduzem código para *bytecode* geram programas mais eficientes do que linguagens interpretadas diretamente. A JVM é uma máquina virtual extremamente otimizada, o que garante um melhor desempenho das aplicações. Como este sistema é projetado para o uso por centenas ou mesmo milhares de funcionários em uma empresa e poucos usá-lo-ão simultaneamente, então este definitivamente não é um problema. Pense-se sobre o número de usuários do GitHub, que é uma plataforma escrita em Ruby on Rails e é hoje o maior repositório de códigos fonte aberto na Internet.

As vantagens do uso de Ruby são as seguintes.

- Existe um tutorial oficial da linguagem, além dos livros e tutoriais alternativos;
- A documentação da linguagem é mais completa e mais clara;
- Existe o Rails, que permite usar o estilo arquitetural Rails;
- Quase todo o uso de Ruby no mercado é voltado para a Internet, enquanto Java é mais dividido;
- A linguagem é mais fácil de ser lida e entendida, o que facilita a manutenção;
- Geração de código com o Rails;
- *Convention over configuration* é uma das convenções de Ruby, o que também ajuda o entendimento do código e facilita o desenvolvimento e a manutenção;
- Rails dá suporte nativo a AJAX (*Asynchronous* Javascript *and* XML, Javascript Assíncrono e XML), Jquery e CoffeeScript, que são ferramentas para facilitar o uso de Javascript, XML (*eXtensible Markup Language*, Linguagem de Marcação Extensível) e JSON (JavaScript *Object Notation*, Notação de Objeto em JavaScript), com diversas funções para tornar páginas da Internet mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações;
- Ferramentas de teste de aceitação mais bem feitas.

Dando preferência para velocidade de desenvolvimento e para manutenção futura por terceiros a escolha por Ruby é a mais sensata.

4.3 Arquitetura e projeto

O projeto do sistema foi dividido em duas partes, a primeira foi o desenvolvimento do modelo de dados, a segunda a arquitetura geral do sistema, bem como o porquê de certas decisões arquiteturais.

4.3.1 Aplicação das definições

A partir das definições apresentadas no item 2.1, a tabela 4.3.1 relaciona as classes programadas no sistema aos conceitos.

Classe	**Conceito**
Ability	Habilidade
KnowledgeArea	Área de conhecimento
CompetenceMatrix	Matriz de competência
Value	Grau de competência

Tabela 4.3.1: *Relação entre as classes do projeto e os conceitos utilizados*

4.3.2 Modelo conceitual de dados

O modelo de dados foi produzido inicialmente em conjunto com uma das atividades da matéria de banco de dados, MAC0426, no primeiro semestre de 2014. No segundo semestre ele foi modificado, influenciado pela documentação do projeto da PRODESP (V. item 3.1.1).
O resultado foi um modelo mais claro e simples como pode ser visto no diagrama ER (Entidades e

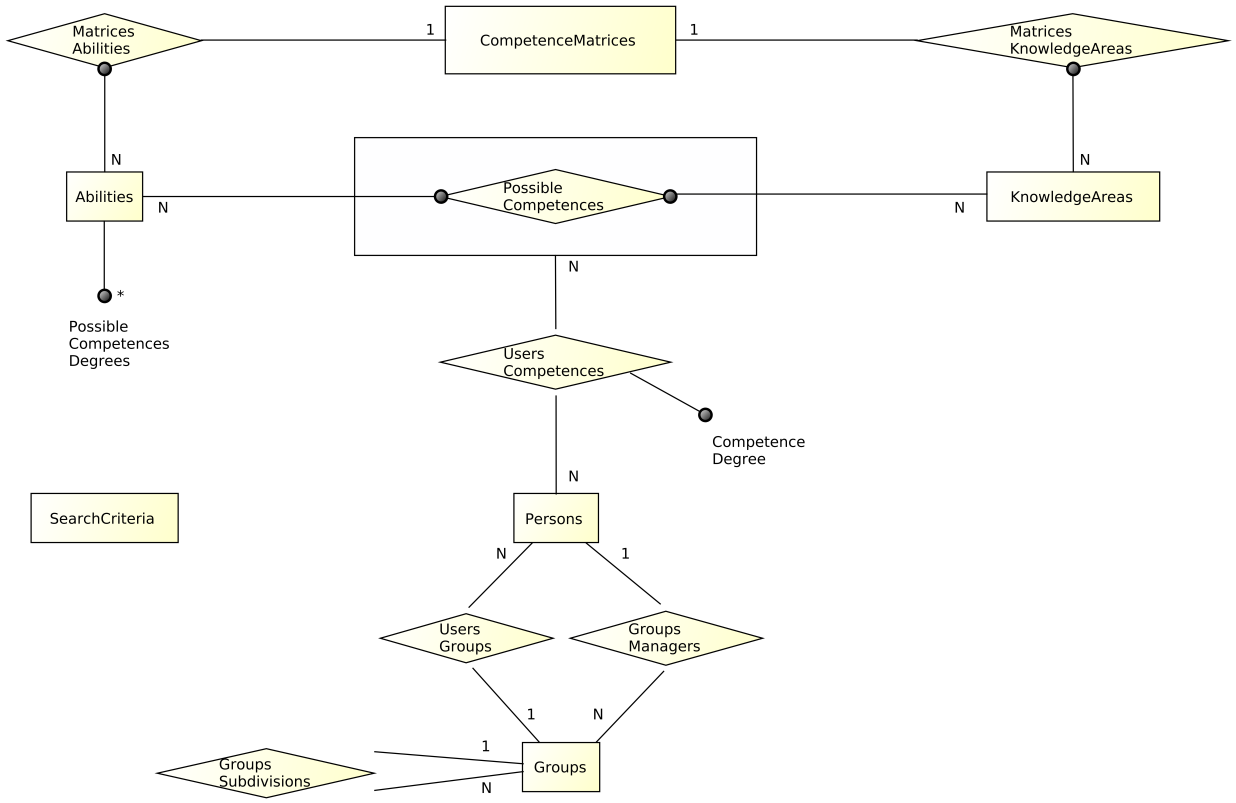


Figura 4.3.1: *Modelo conceitual de dados*

Relacionamentos) com poucos atributos na figura 4.3.1, seguindo padrão de diagramas conceituais de Setzer e da Silva [2005].
Essa figura mostra como ficaram organizadas as entidades e relacionamentos da aplicação. Nessa implementação, uma matriz de competência possui duas listas, uma de habilidades e outra de áreas de conhecimento; um elemento de CompetenceDegrees seria um possível grau que uma competência pode assumir; uma habilidade possui uma lista de graus possíveis; uma competência é uma agregação (segundo em Setzer e da Silva [2005]) que liga habilidade, área de conhecimento, grau e uma pessoa; Um usuário do sistema pode participar de um grupo e um grupo possui um gerente. Além disso, um grupo pode ter vários subgrupos. No caso desse sistema a profundidade máxima de um subgrupo é ilimitada, isto é, existem grupos, subgrupos, subsubgrupos e assim por diante. Uma futura extensão deveria tornar isso configurável por um menu acessível pelo administrador do sistema.

4.3.3 Estilo arquitetural Rails

Seguindo a definição de estilo arquitetural de Taylor *et al.* [2009], o padrão arquitetural do sistema é o Rails, pois escolher usar Rails é escolher o uso de um conjunto de decisões arquiteturais. Essas decisões incluem:

- **Baseado na Internet**

Com Rails o projeto é voltado para a Internet. Rails faz toda a parte de criação de rotas, redirecionamentos e renderização, além de prover uma linguagem de marcação baseada no HTML (*HyperText Markup Language*, Linguagem de Marcação de Hipertexto), chamada ERB (*Embedded* Ruby, Ruby Embutido), para a criação de páginas.

- **Padrão de projeto Active Record**

O padrão de projeto Active Record foi citado pela primeira vez em Fowler [2002] e é o padrão usado para uso de banco de dados com Rails. O padrão usa a classe extra ActiveRecord que funciona com um invólucro de banco de dados orientado a objetos (Kim [1990]) para um banco de dados relacional. Uma instância de uma classe entidade herda de ActiveRecord e representa uma linha da tabela. O ActiveRecord implementa métodos de acesso como *save*. A diferença para o padrão de projeto DAO (*Data Access Object*, Objeto de Acesso a Dados, Oracle [2014]) é que no DAO cada tabela é representada por uma classe chamada entidade e para cada entidade há uma classe com os métodos para o acesso aos dados da entidade enquanto no *ActiveRecord* a própria entidade faz as operações de acesso herdando os métodos da classe *ActiveRecord*.

- **MVC Model 2**

Também conhecido como MVC2, Model 2 e Rails MVC, o MVC Model 2 é um dos padrões semelhantes ao MVC tradicional, que pode ser encontrado em qualquer livro de engenharia de *software*, como Sommerville [2004]. A diferença entre MVC e MVC2 é o fato de que no MVC, um *model* (entidade) pode notificar a *view* (telas, clientes etc) sobre mudanças através do padrão de projeto Observer, e no MVC2 isso não acontece.

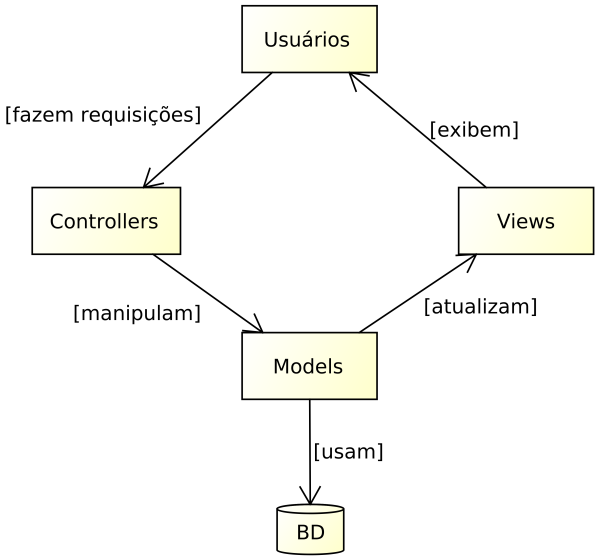


Figura 4.3.2: *Diagrama do padrão arquitetural model-view-controller*

O MVC tradicional funciona como na figura 4.3.2. O usuário faz requisições ao *controller* (classes de negócio); este manipula o que for necessário no *model* e o próprio *model* notifica a *view* sobre mudanças. Em seguida, a *view* exibe ao usuário as mudanças. No caso do MVC2, no lugar do *model* notificar as mudanças, o próprio *controller*, que manipulou o *model*, notifica a *view*, como mostra a figura 4.3.3. A vantagem do Model2 é que a parte de negócios do sistema fica mais concentrada nos *controllers*.

- **Padrão arquitetural RESTful**

REST (*Representational State Transfer*, Transferência de Representação de Estado) é uma técnica de engenharia de *software* para sistemas hipermídia distribuídos, como a Internet. O REST baseia-se em uma série de princípios, como protocolo cliente-servidor sem estado, um conjunto de métodos bem definidos que se aplicam a todos os recursos de informação (no caso do HTTP, *HyperText Transfer Protocol*, Protocolo de Transferência de Hipertexto, sendo os principais *get*, *put*,

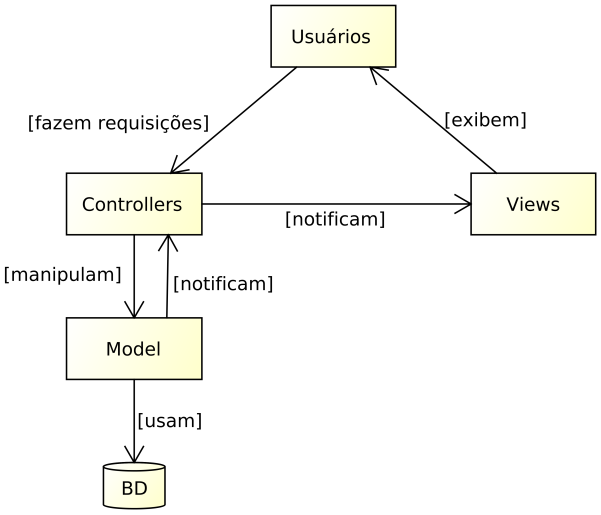


Figura 4.3.3: *Diagrama do padrão arquitetural Model 2*

post e *delete*), uma sintaxe universal, no caso, URIs (*Uniform Resource Identifier*, Identificador Uniforme de Recursos), e o uso de hipermídia, no caso HTML.
Sistemas que aplicam uma interface REST são ditos seguirem uma arquitetura RESTful.

- **Geração de código**

Por meio de descrições em comando de terminal é possível gerar códigos simples em Rails. A vantagem do Rails é que ele cuida da parte 'suja' do trabalho, como criar rotas, cuidar do banco de dados, e criar os arquivos das classes a serem usadas pelo desenvolvedor já com uma descrição mínima, sobrando apenas criar a lógica e a interface.
O comando abaixo cria um novo projeto chamado *syscoma*, organiza as pastas e cria diversos arquivos para o projeto ser iniciado.

```
$ rails new syscoma
```

Outro comando usado foi para criar *controllers* isolados, as suas rotas e as *views* específicas dessas rotas. No exemplo cria-se um *controller* chamado *welcome* com rotas *index* e *index2*. Isso também cria as *views index* e *index2*. Outra vantagem de criá-las por meio de um comando é que ele já cria os arquivos para testes, só faltando escrever os testes em si.

```
$ rails generate controller welcome index index2
```

O seguinte comando cria um *model* com os atributos que o seguem e, assim como no caso do *controller*, ele também cria os arquivos de teste. A diferença para o comando anterior é que ele não cria rotas, mas cria o modelo simples do banco de dados, incluindo a chave estrangeira para *Ability*.

```
$ rails generate model Value rank:integer name:string ability:references
```

Um comando mais abrangente cria um *model* e um *controller* com o mesmo nome. O comando abaixo cria um *model* exatamente como o anterior e ainda cria um *controller* chamado *ValueController* com os métodos *index*, *show*, *new*, *edit*, *create*, *update*, *delete*. Assim como os anteriores, este comando cria diversos arquivos para testes com testes vazios para cada método de cada classe e rota criada.

```
$ rails generate scaffold Value rank:integer name:string ability:references
```

- **DRY - *Don't Repeat Yourself***

Outra característica do estilo arquitetural Rails é o princípio de não repetir código. Em várias partes do sistema há trechos onde são evitadas repetições, p. ex., para evitar a repetição de código

nas *views* há suporte a porções parciais (*partials*) e *templates* de código HTML, que o desenvolvedor pode incluir livremente. Outro exemplo é o uso do ActiveRecord, que evita repetição de métodos de acesso a banco de dados.

- **Convenções sobre configurações**

Outro princípio do estilo Rails são as convenções, p. ex., há uma única convenção de nomes de métodos e classes no código; há também uma arquitetura padrão que é aconselhável que seja seguida. Basicamente todos os itens anteriores desta subseção seguem esse princípio.

Um exemplo para mostrar que a influência do DRY e das convenções no código é bem grande é a classe *CompetenceMatrix* exemplificada a seguir.

```
1 class CompetenceMatrix < ActiveRecord::Base
2   validates :name, presence: true, length: { minimum: 4 }, uniqueness: true
3   has_many :abilities
4   has_many :knowledge_areas
5   attr_accessor :description
6 end
```

O equivalente em Java seria

```
1 class CompetenceMatrix extends ActiveRecord
2   private ArrayList<Ability> abilities;
3   private ArrayList<KnowledgeArea> knowledgeAreas;
4   private String name;
5   private String description;
6   public CompetenceMatrix(String name) throws Exception {
7     if(name==null || name.lenght() < 4 || CompetenceMatrix.find_by("name", name)
          != null)) {
8       throw Exception; //lança alguma exceção de Java
9     }
10  }
11  public ArrayList<Ability> getAbilities(){
12    return abilities;
13  }
14  public void setAbilities(ArrayList<Ability> abilities) {
15    this.abilities = abilities;
16  }
17  public ArrayList<KnowldgeArea> getKnowledgeAreas(){
18    return knowledgeAreas;
19  }
20  public void setKnowledgeAreas(ArrayList<KnowledgeArea> knowledgeAreas) {
21    this.knowldgeAreas = knowldgeAreas;
22  }
23  public String getName(){
24    return name;
25  }
26  public void setName(String name) {
27    this.name = name;
28  }
29  public String getDescription() {
30    return description;
31  }
32  public void setDescription(String description) {
33    this.description = description;
34  }
35 }
```

4.4 Testes

Foi usada a terminologia de testes encontrada em Hansson [2014c], e as encontradas em Chelimsky *et al.* [2010]. Para este projeto foram feitos testes unitários, funcionais, de integração e de aceitação.
Em adição, para testes unitários e testes de sistema foram usados objetos 'dublê' *mock objects*, como definidos em Sommerville [2004].
No caso dos testes de unidade, os testes funcionais e os testes de integração são suportados por padrão pelo Rails. Apesar disso foi utilizado o arcabouço de testes Rspec, pois testes escritos neste arcabouço resultam em testes mais legíveis que os da biblioteca padrão. Os testes podem ser vistos na pasta *spec* na raiz dos diretórios do sistema. Os testes de unidade no Rails testam apenas os *models*. Para os *controllers* usam-se testes de sistema chamados testes funcionais. A diferença dos dois é que os testes de *models* testam além do método em si o estado final do objeto após o método, enquanto os funcionais testam além da saída do método o redirecionamento para a *view* específica e a existência e o estado dos elementos na *view*.
Os testes de integração testam 'passeios' por funcionalidades do sistema, testando em conjunto vários *controllers*, *models* e *views*.
Os testes de aceitação são histórias que descrevem em linguagem quase natural o que o sistema deve fazer. A diferença para os testes de integração, além da linguagem, é que estes realmente representam um usuário comum, fazendo as ações de fato. Este tipo de teste não é suportado nativamente pelo Rails. Neste projeto foi usado o arcabouço Cucumber (Hellesøy *et al.* [2014]).

4.4.1 Fixtures e FactoryGirl

O padrão no Rails para o uso de *mock objects*, isto é, instâncias dos models usados nos testes, são as *fixtures*. A sintaxe para a escrita deles é YAML (*YAML Ain't Markup Language*, YAML não é linguagem de marcação), que é exemplificada na seção 4.5.
Neste projeto, ao invés de *fixtures*, foram usadas *factories* com o arcabouço FactoryGirl. O uso de *fixtures* torna difícil ver causa e efeito, já que tudo é feito em um contexto separado da aplicação. Por outro lado, *factories* colocam a lógica toda no teste, com isso o código fica mais elegante e o uso é mais flexível. Outra vantagem é que a sintaxe é a do próprio Ruby, tornando o texto mais legível. Dois exemplos podem ser vistos a seguir.

```
1 #Arquivo spec/factories/users.rb
2 FactoryGirl.define do
3   factory :manager, class: User do
4     email "manager@company.com"
5     name "Foo Bar"
6     password "12345asd"
7     password_confirmation "12345asd"
8     admin false
9   end
10  factory :employee, class: User do
11    email "employee@company.com"
12    name "John Doe"
13    password "12345asd"
14    password_confirmation "12345asd"
15    admin false
16  end
17  factory :admin, class: User do
18    email "admin@company.com"
19    name "administrator of the system"
20    password "12345asd"
21    password_confirmation "12345asd"
22    admin true
23  end
24 end
```

```
1 # Arquivo spec/factories/groups.rb
2 FactoryGirl.define do
3   factory :group, aliases: [:subgroup_of] do
4     name "A commom group"
5     subgroup_of nil
6     manager
7   end
8   factory :subgroup, class: Group do
9     name "A commom subgroup"
10    subgroup_of
11    manager
12  end
13 end
```

E para usar isso nos testes é só instanciar como descrito abaixo:

```
1 build(:user) #cria uma instância de User como definido no arquivo spec/factories
    /users.rb.
```

4.4.2 Verificação de cobertura de testes

Para a verificação de cobertura de testes foi usado o arcabouço *SimpleCov* (Olszowka [2014]) que gera um relatório dos testes com a análise como mostra a figura 4.4.1. Acionando em qualquer um dos *links* aparece uma tela como na figura 4.4.2, onde as linhas verdes são as linhas pelas quais os testes passaram, as linhas vermelhas são as linhas pelas quais os testes não passaram e o número à direita é o número de vezes que a linha foi executada.

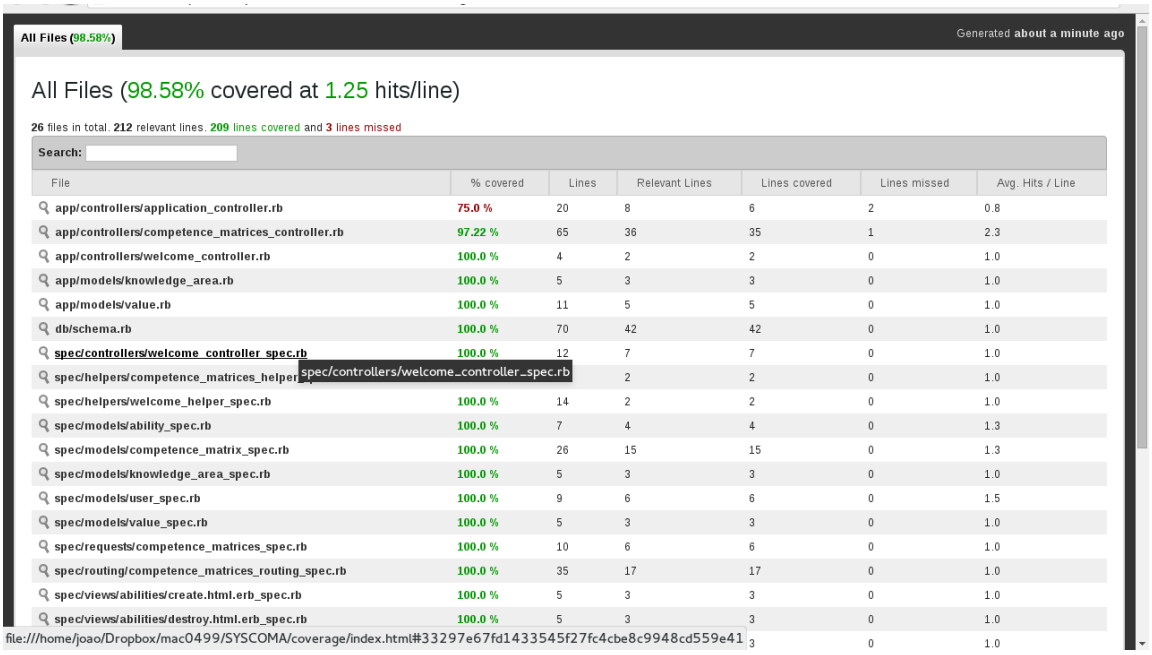


Figura 4.4.1: *Tela da análise de cobertura de testes*

4.5 Internacionalização

Uma das vantagens de usar Rails é de ele ter já implementada uma API de internacionalização. O exemplo de arquivo de internacionalização é a língua padrão, no caso inglês. Assim como as *fixtures*, os arquivos de internacionalização usam o formato YAML. Os arquivos de internacionalização podem ser encontrados em *config/locales/*, havendo um arquivo para cada língua. No momento há

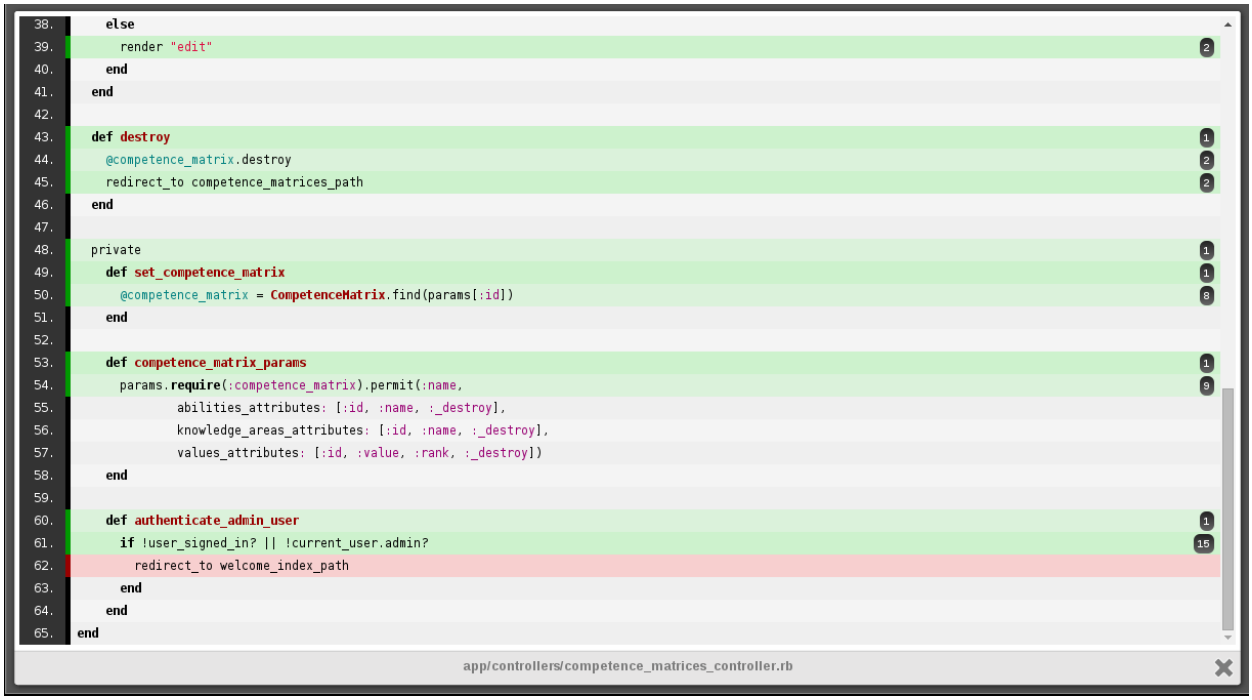


Figura 4.4.2: *Tela da análise de cobertura de testes de uma classe*

apenas português e inglês. Para adicionar uma nova língua é só copiar um dos arquivos que já existem e traduzir seguindo a sintaxe. O exemplo a seguir é uma amostra de como fica um arquivo de tradução.

```
1 # Um possível en.yml em config/locales
2 en:
3   links:
4     edit: "Edit"
5     destroy: "Delete"
6   title: "SYSCOMA"
```

```
1 # Um possível en.yml em config/locales
2 pt:
3   links:
4     edit: "Editar"
5     destroy: "Remover"
6   title: "SYSCOMA"
```

Uma característica dos arquivos *.yml* é que o alinhamento vertical é importante. Sendo assim, para usar uma palavra desse dicionário é só usar a função *t* do módulo I18n, como exemplificado abaixo. O I18n também permite variações nos vocabulários entre diferentes empresas. Por exemplo se no lugar de *persons* a empresa quiser que apareça *people* basta alterar nos arquivos *config/locales/*.

```
I18n.t('links.edit') # equivale a "Edit" se estiver em inglês
I18n.t('title') # equivale a "SYSCOMA"
```

A língua padrão está definida no arquivo *config/locale.rb* no trecho "I18n.default_locale = :en". Trocando ":en" por ":'pt-BR'" como descrito nos comentários do arquivo é possível verificar o sistema todo em português. Uma tarefa futura seria fazer essa transição por meio do próprio sistema e tornar configurável para cada usuário a escolha da língua de preferência, sem precisar alterar diretamente o código.

4.6 Uso da biblioteca Devise

Criar sistema de sessões e senhas requer muito cuidado, em especial para evitar ataques e roubos de senhas. Para isso, é importante que tudo seja criptografado. É importante que sejam salvos registros de acessos e atividades, e também que seja possível bloquear acessos após um número de tentativas de acesso com falha, para evitar ataque de tentativa e erro.
Para simplificar todo o processo foi usada uma biblioteca que facilita todo esse trabalho, o Devise, que cuida da autenticação do sistema. O seu uso permite fazer o acesso a variáveis que indicam se o usuário está numa sessão (*logged in*) e quais usuários estão com sessão aberta. Para isso há um objeto que é a representação do usuário. O uso dela é bem simples. O comando a seguir é tudo que é necessário para a instalação após adicioná-la nas bibliotecas do sistema.

```
$ rails generate devise:install
```

Em seguida, para criar um *model* com criação de sessões e todos os artifícios de autenticação e criptografia basta executar o comando seguinte.

```
$ rails generate devise User
```

Esse comando cria uma classe User e os *controllers* e *views* necessários para a autenticação. Para uma futura personalização da aplicação é possível adicionar Omniauth, que é a autenticação no sistema a partir do Facebook, Google ou outras plataformas com entradas compatíveis.
Com o Devise devidamente instalado é possível receber o usuário da sessão atual a partir do método *current_user* dentro dos *controllers* e das *views*. Também é possível verificar se o usuário criou uma sessão com o método *signed_in?*.

4.7 Mais sobre Ruby on Rails

Todos os conceitos usados aqui que se referem à linguagem Ruby podem ser encontrados em Matsumoto [2014]. Os conceitos que se referem ao arcabouço Rails podem ser encontrados em Hansson [2014a] e Hansson [2014b].

4.8 Código

Todo o código escrito foi feito tendo em mente alguns princípios de boa prática de programação. Em relação ao estilo, em primeiro lugar foram levados em consideração os princípios expostos em Martin [2008], como nomes de classes e métodos, tamanho de classes e métodos, acoplamento e outros conceitos. Também foram levados em conta os princípios da própria linguagem Ruby e do arcabouço Rails e os princípios estabelecidos para o próprio projeto, descritos em https://github.com/SYSCOMA/SYSCOMA/wiki/Standards.
Quanto ao fluxo de trabalho foram mantidos em mente sempre dois princípios de métodos ágeis, como os testes (TDD, *Test-Driven Development*, Desenvolvimento Dirigido a Testes) e *baby steps*, descritos a seguir.
Um breve resumo de TDD seria a atividade de criar primeiramente os testes, vê-los falhar e depois implementar o código que passa no teste.
Um breve resumo de *baby steps* seria o ato de dividir as mudanças no código em pequenas evoluções usando um controle de versões, no caso, Git.

4.9 Resultado

No sistema há 3 categorias de usuário, podendo um usuário pertencer a mais de uma. Administrador do sistema, ou apenas administrador é quem cuida da parte de configuração do sistema, de criação e edição de definições de matrizes, cadastro de usuário etc. Gerente é quem coordena um ou mais grupos, podendo remover e inserir pessoas no grupo. Funcionário é qualquer pessoa do

sistema, sendo que qualquer pessoa cadastrada como usuário com senha pode fazer uma busca por pessoas que têm determinadas competências.
A primeira tela a ser exibida ao entrar no sistema, caso o usuário não esteja em uma sessão, é uma tela simples e sem nada como na figura 4.9.1.

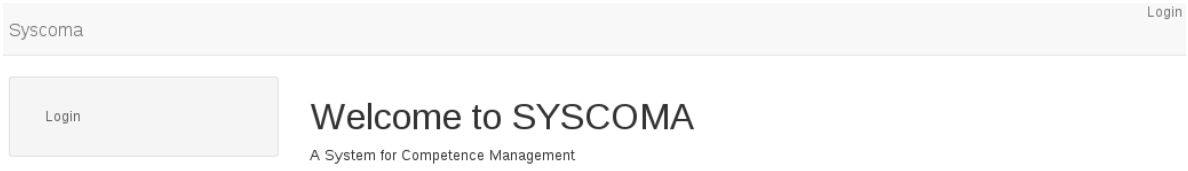


Figura 4.9.1: *Tela de entrada*

Caso o usuário deseje usar o sistema aciona *Login* e aparecerá a tela 4.9.2.

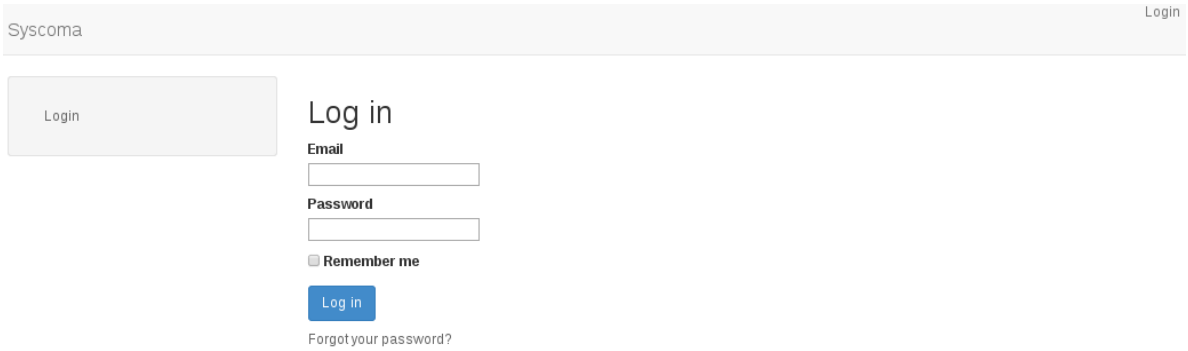


Figura 4.9.2: *Tela de nova sessão*

Criada a sessão o usuário é redirecionado para a tela seguinte. No caso do usuário ser um administrador ele é redirecionado para a tela 4.9.3. Um administrador tem poderes especiais no sistema, podendo adicionar, alterar e remover outros usuários, criar, alterar e remover grupos, procurar usuário diretamente pelo nome, inserir, alterar e remover as definições das matrizes de competência. Outros usuários não têm o poder de criar, editar ou remover as definições de matrizes, apenas preencher e alterar suas próprias, nem criar, alterar ou remover grupos. Um usuário comum tem sua página inicial como na figura 4.9.4. Caso ele seja gerente de algum grupo, há algumas opções a mais como mostra a figura 4.9.5.

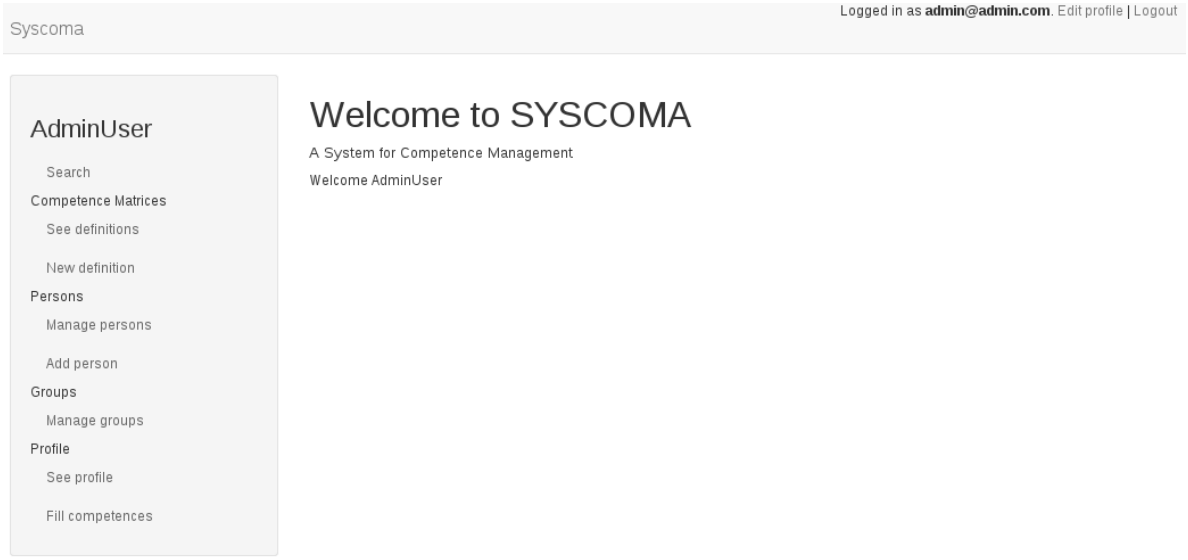


Figura 4.9.3: *Página inicial de um usuário administrador*

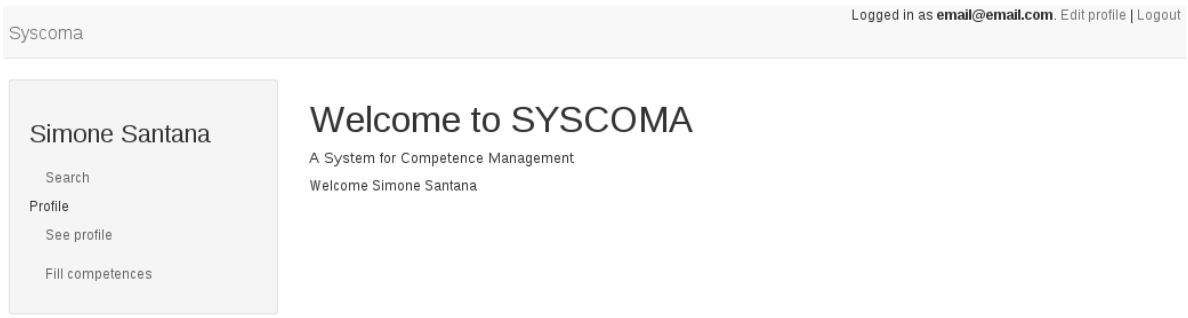


Figura 4.9.4: *Página inicial de um usuário comum*

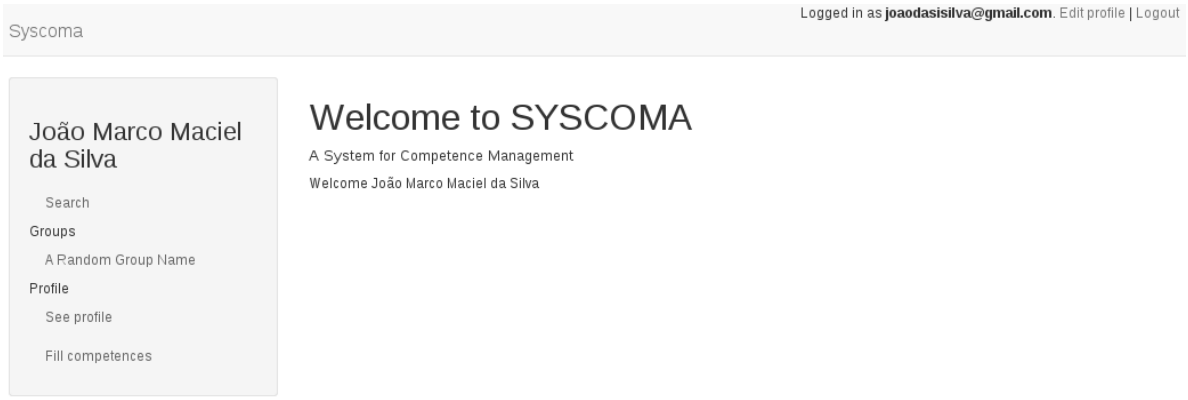


Figura 4.9.5: *Página inicial de um usuário gerente*

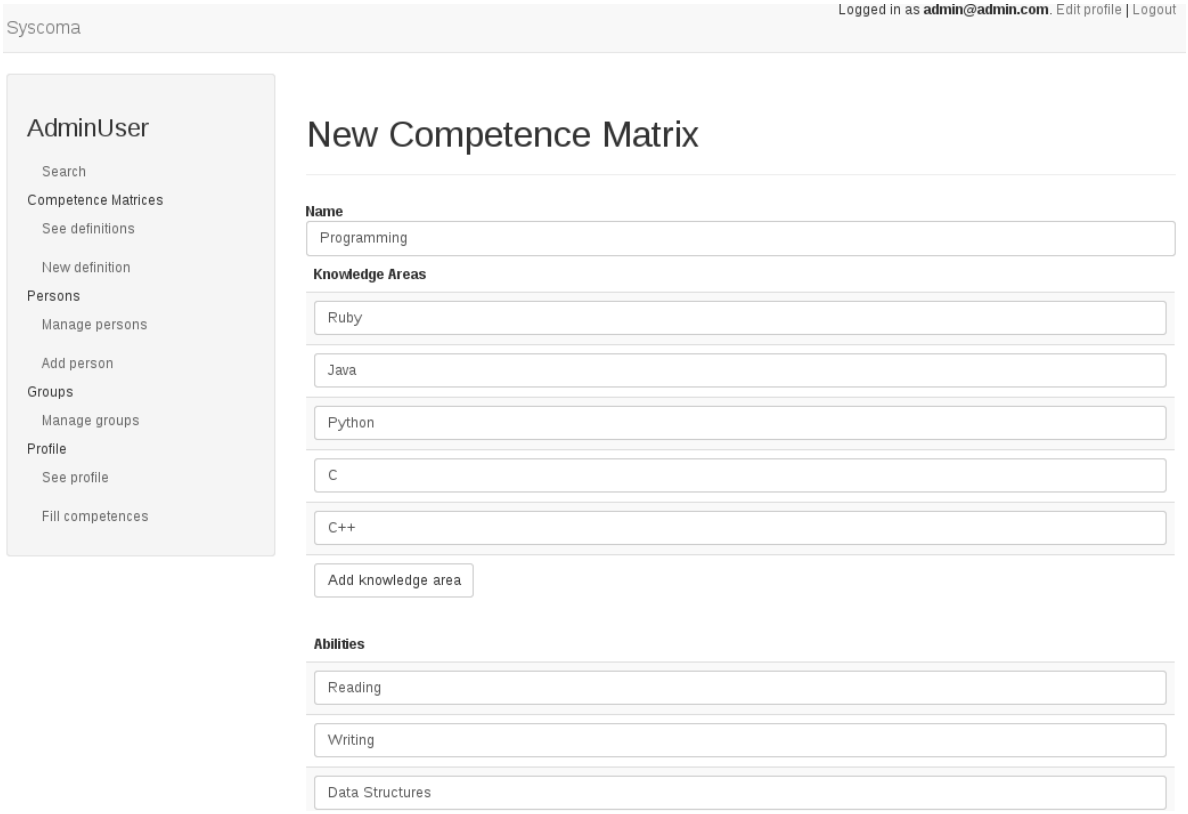


Figura 4.9.6: *Tela de criação e edição de matrizes de competência*

No sistema há duas maneiras de criar matrizes de competência, uma antes de colocar o sistema para executar a primeira vez, por um *script* em Ruby na pasta db em um arquivo chamado *seeds.rb*.

Depois basta executar o comando *$ rake db:seed* no terminal. Esse método também funciona após a aplicação estar sendo usada, mas isso não é recomendado, pois pode causar inconsistência no banco de dados. A outra é pelas telas da aplicação, pois nelas só usuários administradores podem criar, editar ou alterar as definições das matrizes. Da tela de criação consta o nome da nova matriz, uma lista das áreas de conhecimento a serem estabelecidas para a mesma, uma lista das habilidades e uma lista dos graus de competência válidos para todas as habilidades como mostra a figura 4.9.6.

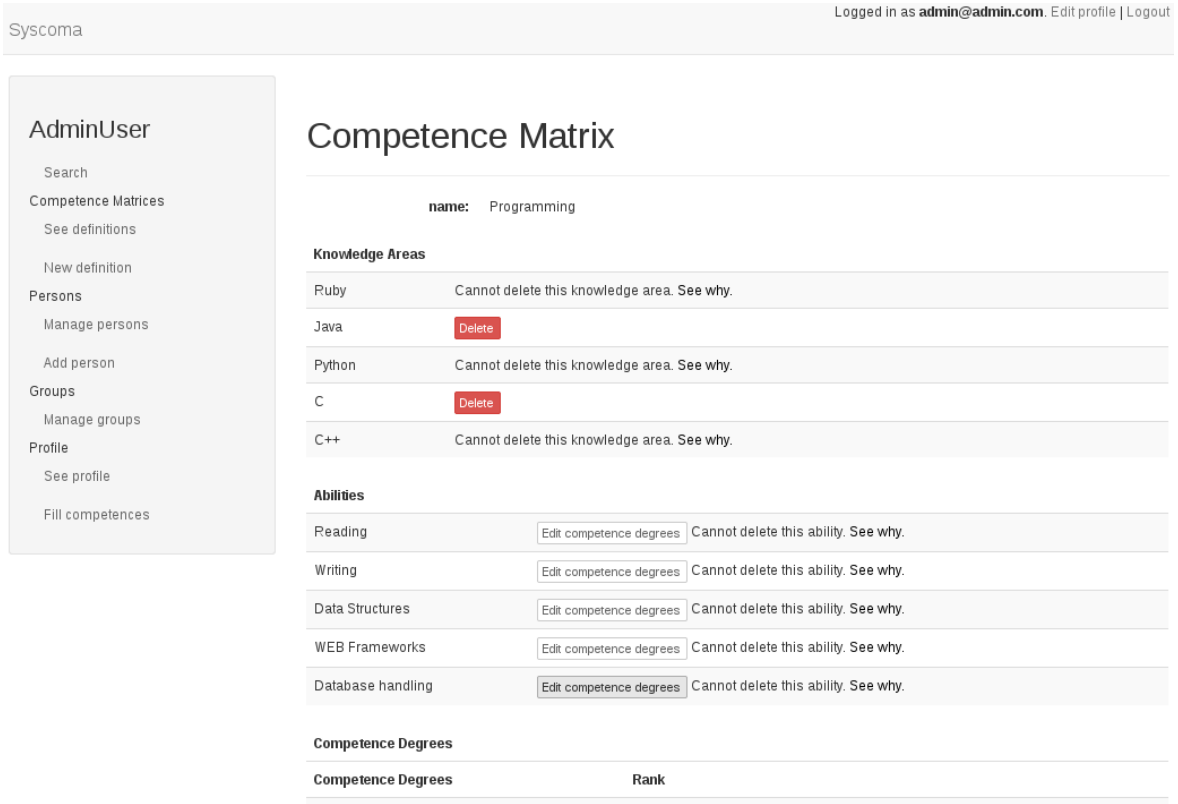


Figura 4.9.7: *Tela de exibição de definição de uma matriz*

Caso seja necessário é possível atribuir uma lista de graus de competência específicos para cada habilidade clicando em *Edit competence values* ao lado da habilidade na tela 4.9.7, seguindo para a tela 4.9.8.

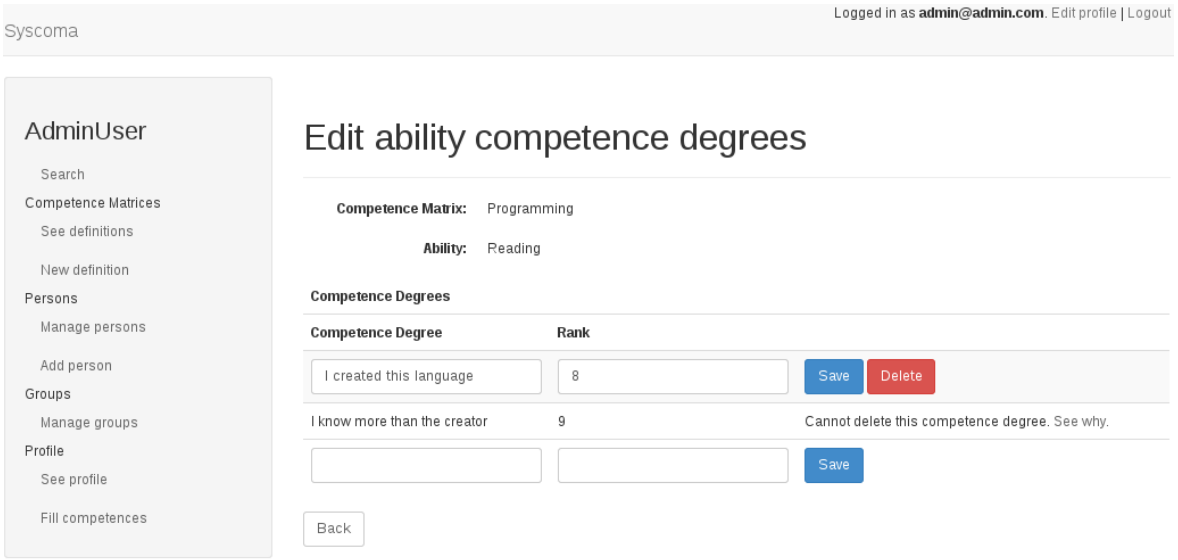


Figura 4.9.8: *Tela de edição de graus de competência das habilidades*

Caso algumas dessas entidades (matrizes, áreas de conhecimento, habilidades ou graus de competência) tiver alguma competência preenchida por alguma pessoa essa entidade não fica disponível para ser removida. Acionando o *link* '*See why.*' disponível é possível ver as pessoas que preencheram tal entidade, como mostra a figura 4.9.9.

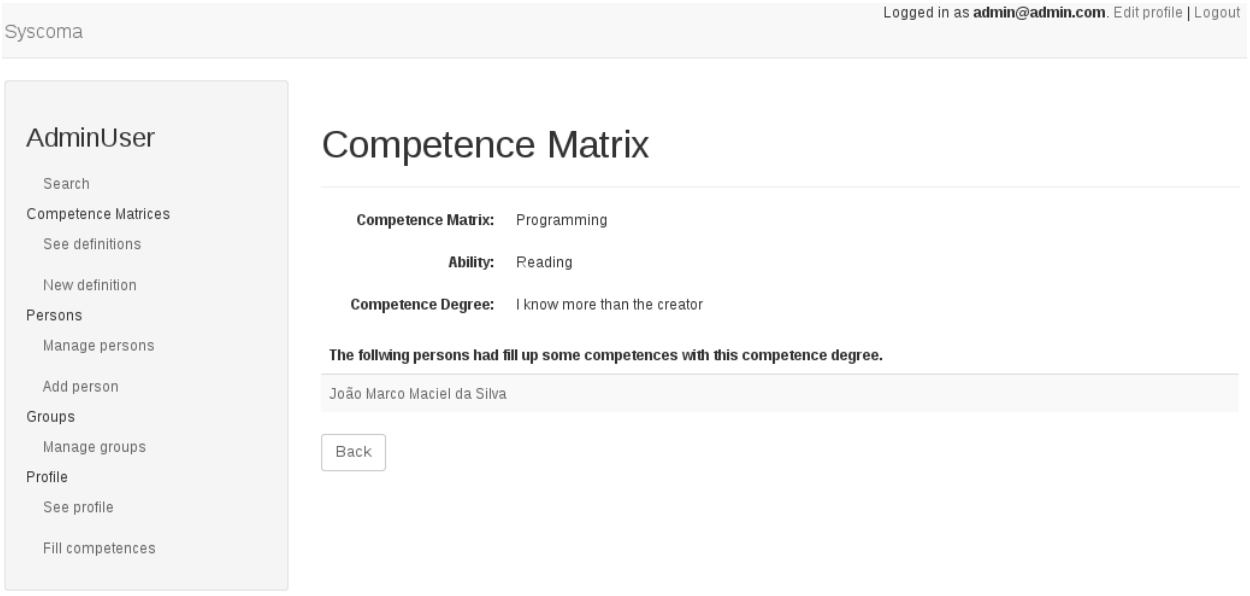


Figura 4.9.9: *Tela com a lista das pessoas que preencheram um certo grau de competência*

Um administrador do sistema ou um gerente de um grupo podem editar quem participa do grupo, bem como incluir e excluir usuários dele, como na figura 4.9.10. Digitando parte de um nome ou um nome completo na caixa *Add employee* da figura 4.9.10 passa-se para a tela 4.9.11, bastando acionar o nome para adicioná-lo ao grupo.

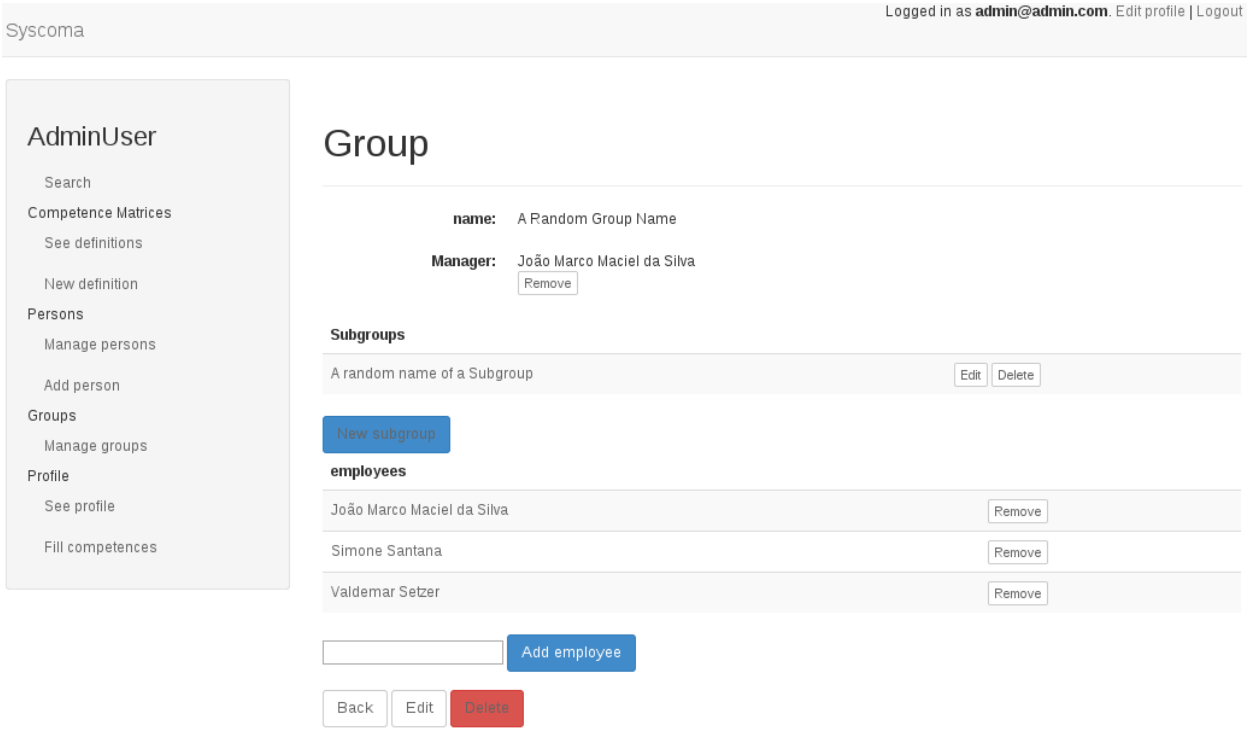


Figura 4.9.10: *Tela de edição de grupo*

Um dos itens mais importantes deste projeto é o preenchimento e a exibição de matrizes. O preen-

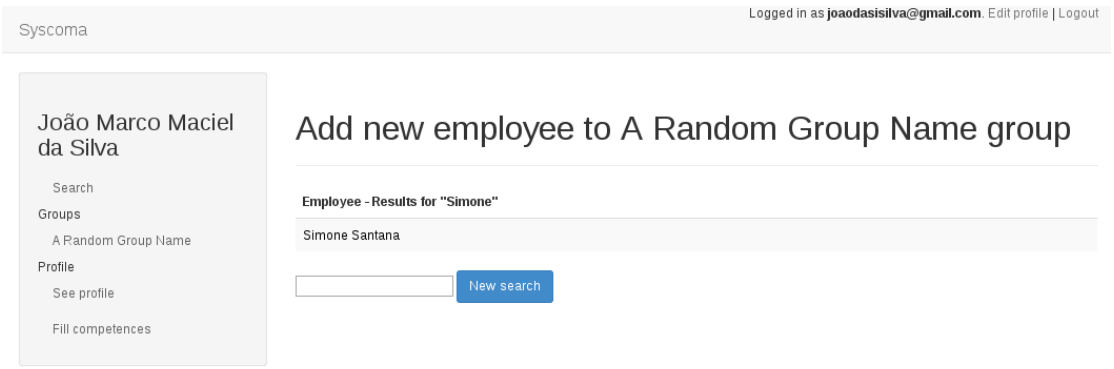


Figura 4.9.11: *Tela com o resultado da busca por funcionários*

chimento é feito em três telas: o usuário primeiro escolhe a matriz (figura 4.9.12), depois a área de conhecimento (figura 4.9.13) e o grau de competência que quer atribuir para cada habilidade (figura 4.9.14). Essas telas só são acessíveis pelos administradores, pelo gerente do grupo do usuário e por ele mesmo.

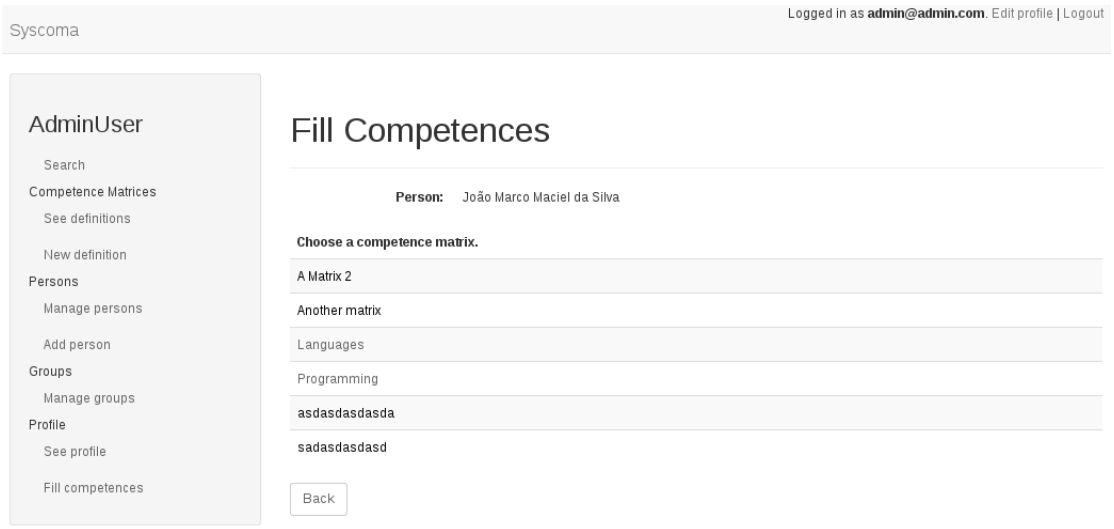


Figura 4.9.12: *Tela de escolha da matriz que deseja preencher*

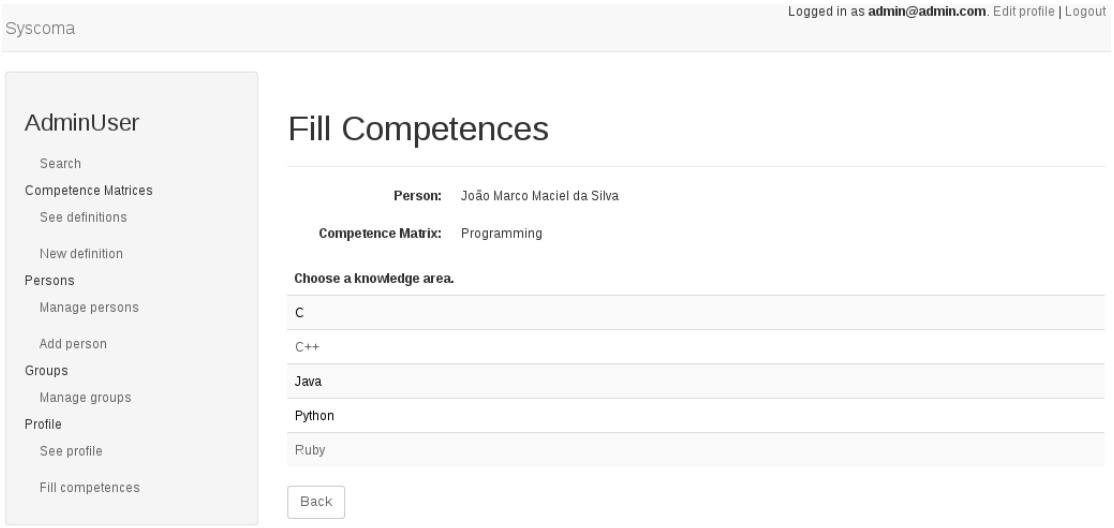


Figura 4.9.13: *Tela de escolha da área de conhecimento*

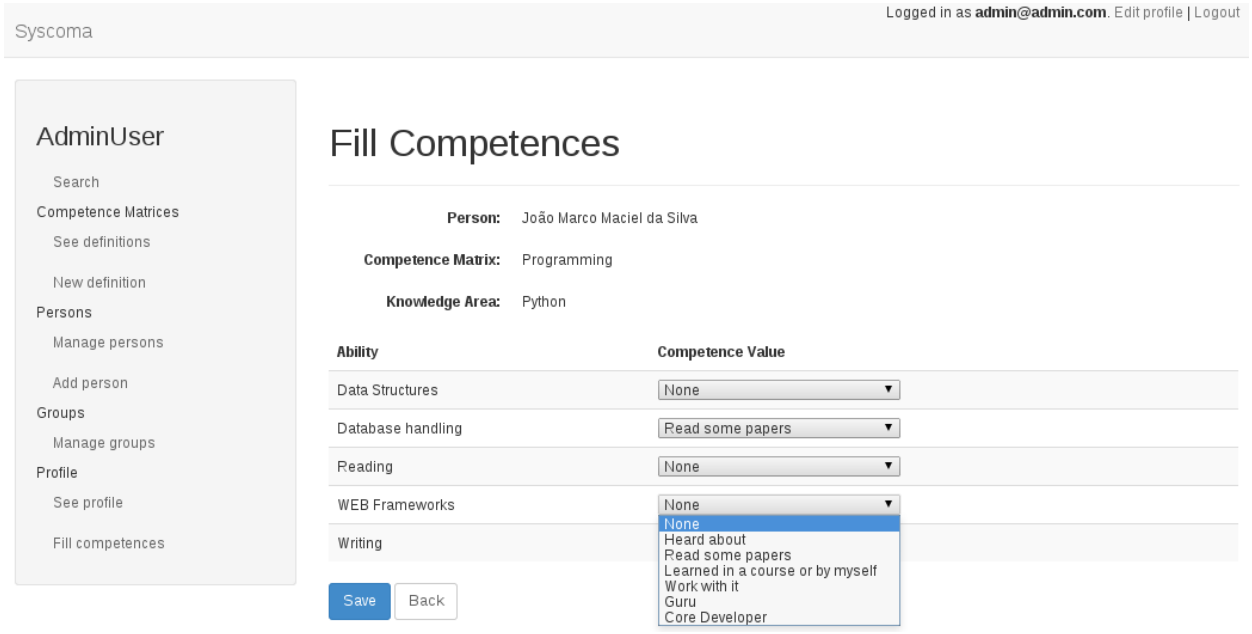


Figura 4.9.14: *Tela de escolha dos graus de competência das habilidades*

Outro item importante é a exibição de matrizes. A primeira tela do perfil do usuário mostra as matrizes que ele preencheu, como exibido na figura 4.9.15. Acionando um dos nomes de matrizes a tela seguinte é semelhante à exibida na figura 4.9.16. Note que nesta figura não há linhas vazias, pois estas são omitidas.

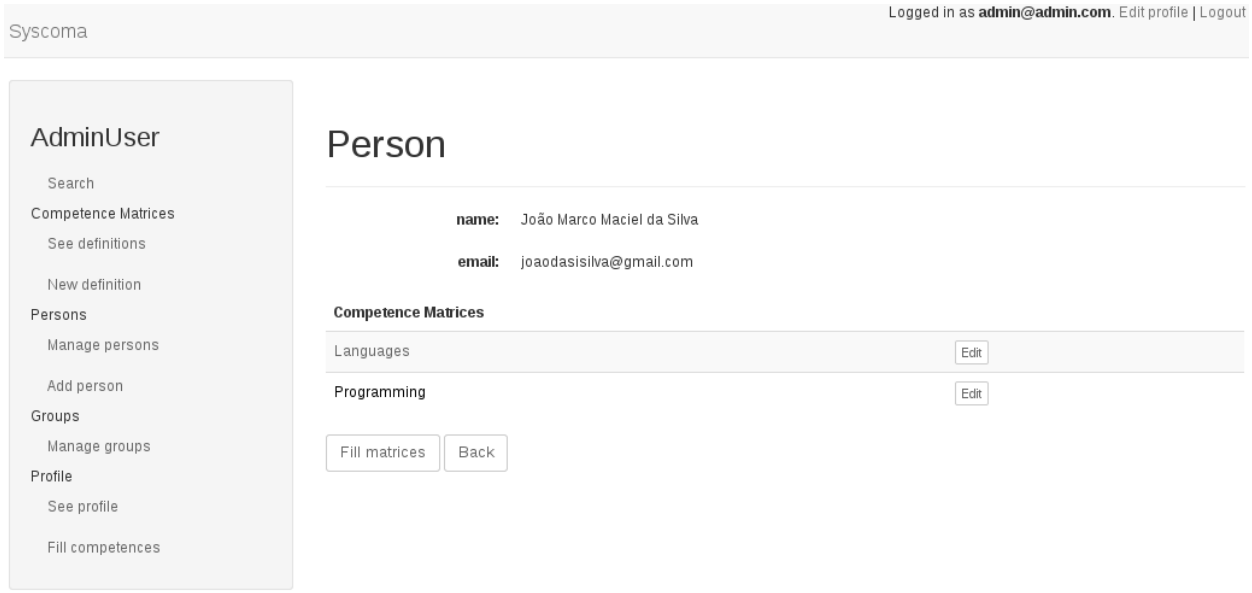


Figura 4.9.15: *Tela de perfil de um usuário*

O último item exibido nesta parte é a busca por pessoas.
Na tela da busca é possível montar a busca de diversas maneiras. Inicialmente, sem buscas armazenadas a tela de busca aparece como na figura 4.9.17.
Seguindo os botões *next*, escolhendo a matriz, a área de conhecimento, a habilidade e o operador de comparação chega-se à tela 4.9.18.
Acionando *Add* chega-se à tela 4.9.19, semelhante à tela 4.9.17, porém com o item *formula* preenchido com o critério adicionado e com uma caixa de escolha de um conectivo (AND ou OR, para combinar critérios de busca) antes da escolha matriz para o próximo critério.
Atribuindo um *label* pode-se associar um nome a qualquer critério de busca, mesmo compostos.

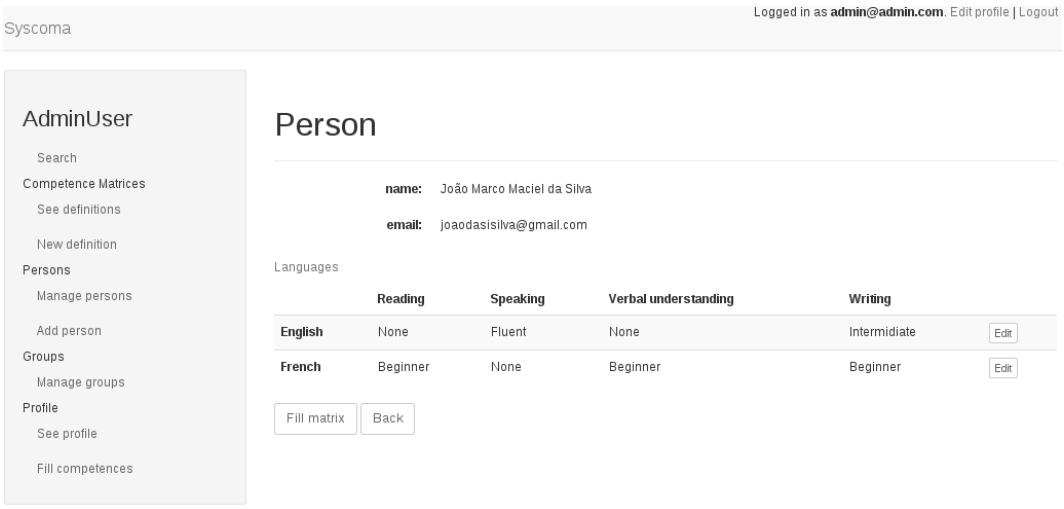


Figura 4.9.16: *Tela de exibição de matriz de competência preenchida*

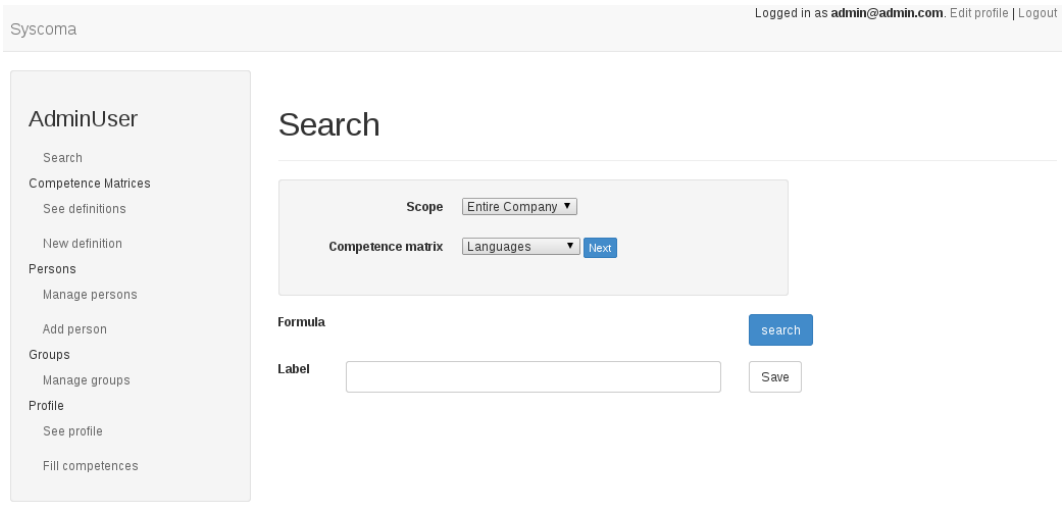


Figura 4.9.17: *Tela de nova busca de pessoas*

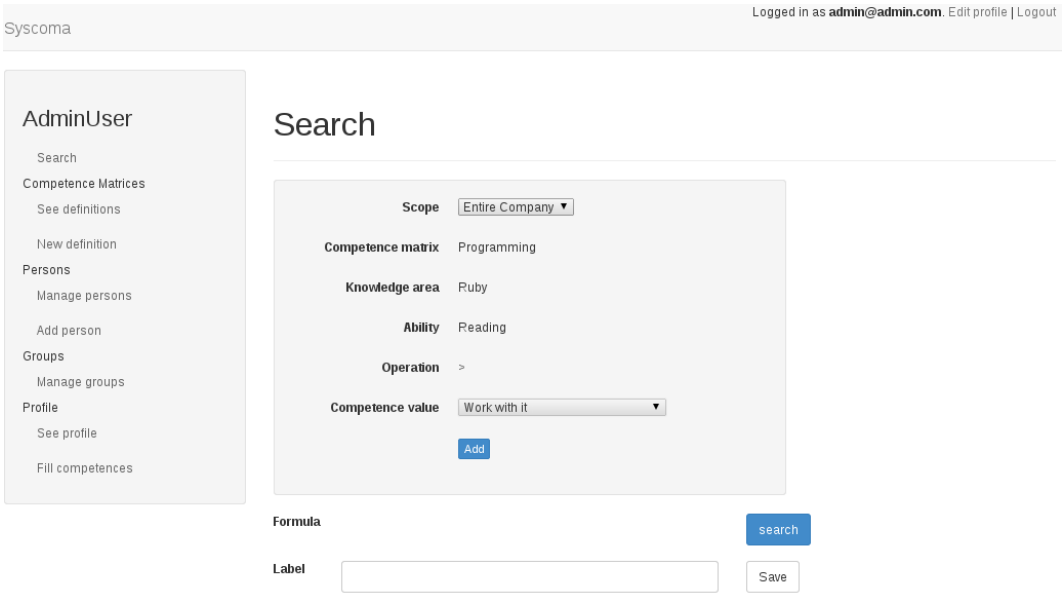


Figura 4.9.18: *Tela de busca de pessoas, escolha do grau de competência*

Acionando-se *save* é possível criar buscas compostas com outras buscas já armazenadas, usando seu nome (*label*), como na figura 4.9.20. Com isso, pode-se estabelecer precedências dentro dos critérios de busca. Nesta tela, acionando *See*, é possível ver qual o critério de busca gerado.

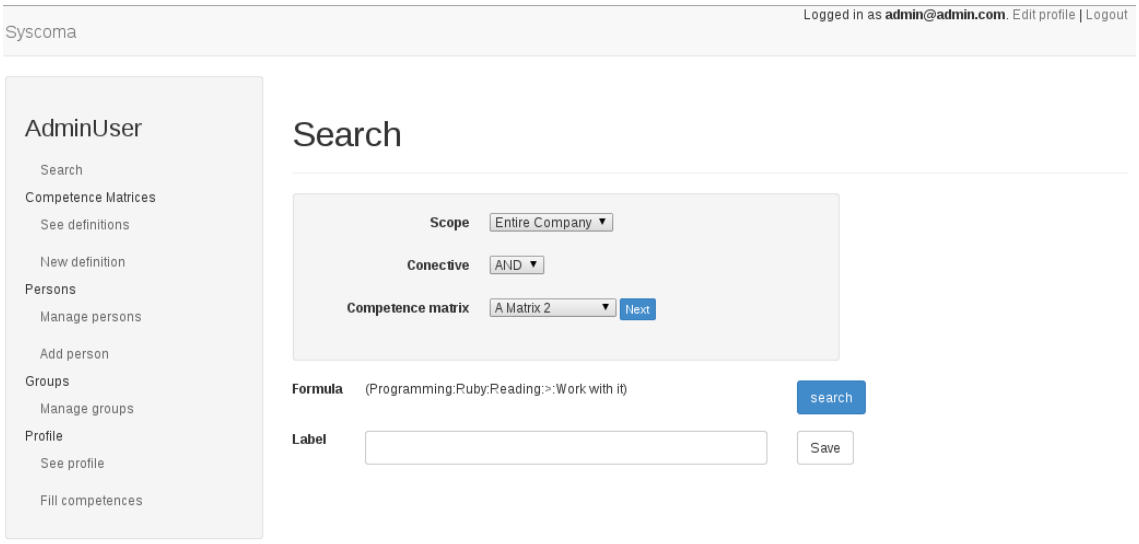


Figura 4.9.19: *Tela de busca de pessoas*

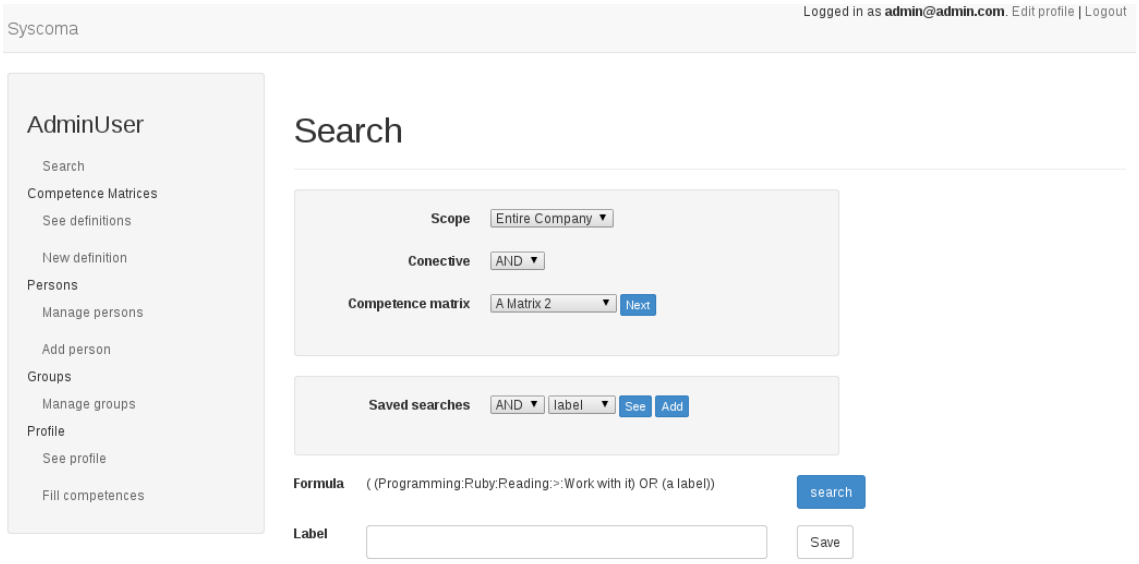


Figura 4.9.20: *Tela de busca de pessoas com buscas armazenadas*

É recomendado que dê-se *labels* que identifiquem cada busca, pois esta aparecerá em *formula* como na figura 4.9.20 apenas como seu *label*.
Note que é possível escolher o escopo da busca na figura 4.9.18 e 4.9.20. Com isso é possível limitar a busca a alguns grupos ou a empresa inteira.
Acionando *Search* chega-se a uma tela como na figura 4.9.21, com a lista dos resultados. Esse resultado é uma lista de *links* para o perfil de cada usuário.
A sintaxe da fórmula que é exibida ao usuário é descrita pela seguinte gramática.

```
<Formula> → <Element> | (<Formula>) AND (<Element>) | (<Formula>) OR (<Element>)
<Element> → <Competence Matrix>:<Knowledge area>:<Ability>:<Operator>:<
    Competence Degree> | <Search label>
<Search label> → <Formula>
```

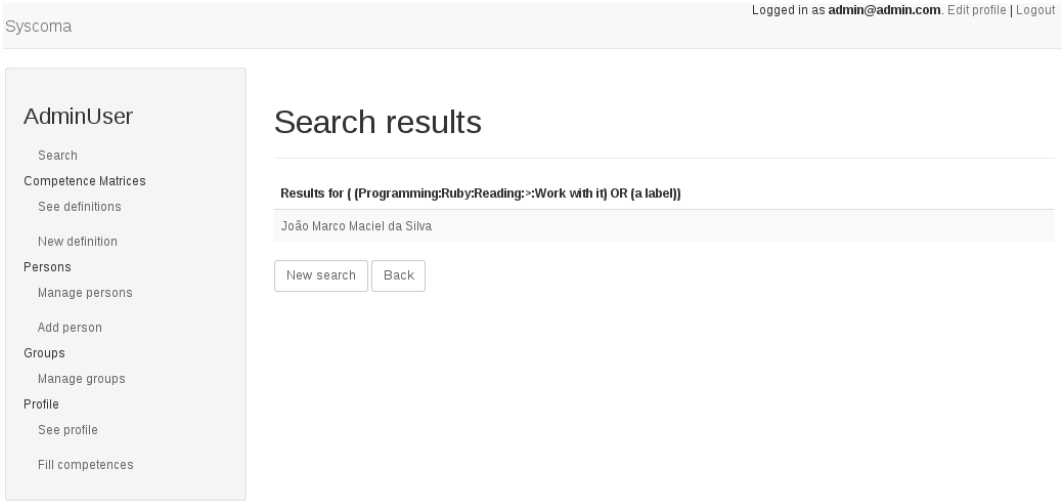


Figura 4.9.21: *Tela com o resultado da busca de pessoas*

Um exemplo disso pode ser visto na tela 4.9.21.
Porém, essa tela mostra o que é exibido pelo sistema, mas não o que é armazenado e interpretado. A exibição serve apenas para que fique fácil para o usuário do sistema reconhecer o critério de busca. A sintaxe efetiva da expressão como usada interiormente pelo sistema é a descrita a seguir.

```
<Formula> → <Element> | (<Formula>)AND(<Element>) | (<Formula>)OR(<Element>)
<Element> → <Knowledge area id>:<Ability id>:<Operator>:<Competence Degree id> |
     <Search id>
<Search id> → <Formula>
```

Dessa forma fica mais fácil programar um interpretador dessa formula. O equivalente interno do exemplo dado na figura 4.9.21 seria algo como *((7:16:>:4)OR(35))* onde *7* seria o identificador da área de conhecimento no banco de dados, *16* seria o identificador da habilidade, *4* seria o identificador do grau de competência (*0* caso fosse grau inexistente), e *35* seria o identificador de uma outra busca armazenada.
Com isso é possível fazer buscas de quem detém certo conhecimento desejado ou mínimo, usando os operadores de comparação $>$ e $>=$. Para se saber quem pode ser um candidato a um treinamento, i. é, detém uma competência abaixo de um mínimo, podem-se usar os operadores $<$ e $<=$.

Capítulo 5

Desenvolvimento Futuro

Para que o projeto fique realmente bom e completo ainda há muito a ser feito.

5.1 O que precisa ser feito

- Testar um caso real de uso, em uma empresa, preenchendo-se matrizes de vários funcionários;
- Tornar configurável o escopo de busca permitido por cada usuário;
- Criar permissões temporárias para os usuários;
- Criar configurações padrão de matrizes de competência para certas áreas, como por exemplo TI, chão de fábrica, indústrias específicas etc;
- Escrever um manual de instruções, semelhante ao que está no arquivo README.rdoc, que está na raiz do projeto, mas incluindo instruções de uso;
- Completar arquivo README.rdoc, que está na raiz do projeto;
- Limitar a profundidade dos subgrupos;
- Escrever testes de aceitação;
- Tornar configurável para cada usuário a língua do sistema que ele deseja usar;
- Há uma inconsistência nas buscas quando se remove uma busca armazenada e que está sendo usada em outra busca. É importante remover essa inconsistência.

5.2 O que é desejável que seja feito

- Múltiplos sistemas. Criar escopos completamente separados para que uma empresa de consultoria e de gerenciamento terceirizado tenha matrizes de competência de funcionários de empresas diferentes no mesmo sistema;
- Currículo. Produzir um relatório com todas as competências cadastradas por um usuário, em forma de currículo;
- Traduções. Inserir outras línguas em *config/locales/*.

5.3 O que pode ser feito

- Uma extensão para buscas de funcionários que precisam de treinamento, registrar o treinamento que eles precisam e lhes enviar *e-mail*. No momento só é possível fazer a busca;
- Deveria ser possível um usuário pertencer a mais de um grupo;
- Integrar com alguma ferramenta ERP;
- Integrar com alguma ferramenta HRM;
- Integrar com alguma ferramenta de gerenciamento por desempenho.

Capítulo 6

Conclusões

Com esse trabalho foi possível treinar diversas técnicas de desenvolvimento, aprender a colocar em produção um sistema baseado na Internet e colocar em prática diversos conceitos sobre computação aprendidos dentro e fora do curso.
O sistema resolve o problema ao qual se propõe e é fácil de ser instalado e usado por empresas e por qualquer pessoa. O código é fácil de ser entendido, modificado e estendido.
Com o uso desse sistema é possível, de modo prático, rápido e fácil, acelerar o processo de seleção interna de funcionários, de busca por habilidades incomuns mas necessárias em uma empresa, de formação de equipes de projeto segundo as competências requeridas, de avaliação de funcionários, de seleção dos funcionários capazes de dar treinamento e dos que o necessitam.

6.1 Considerações finais

Considerando que ainda há bastante a ser feito para este projeto conseguir se manter operante após a conclusão da matéria, o autor compromete-se a manter por pelo menos mais um ano o sistema atualizado e com mais funcionalidades, além de tentar testá-lo em um ambiente real como o CCSL. Desta forma haverá tempo hábil para o projeto ser passado a algum aluno de iniciação científica ou ser usado como um projeto na matéria de MAC0445 (Laboratório de Programação Extrema). O projeto, como foi entregue está na página https://linux.ime.usp.br/~jaodsilv/mac0499 e a versão mais atualizada em https://github.com/SYSCOMA/SYSCOMA/tree/dev.

Referências Bibliográficas

Brandão e Bahry (2014) Hugo Pena Brandão e Carla Patricia Bahry. Gestão por competências: métodos e técnicas para mapeamento de competências. *Revista do Serviço Público*, 56 (2):p–179. Citado na pág. 8

Brandão e Guimarães (2001) Hugo Pena Brandão e T de A Guimarães. Gestão de competências e gestão de desempenho: tecnologias distintas ou instrumentos de um mesmo construto. *RAE*, 41 (1):9. Citado na pág. 8

Brandão *et al.* (2008) Hugo Pena Brandão, Marco Vinício Zimmer, C Pereira, Fernanda Marques, H Costa, Pedro Paulo Carbone e V Almada. Gestão de desempenho por competências: integrando a avaliação 360 graus, o balanced scorecard ea gestão por competências. *Revista de Administração Pública*, 42 (5):875–98. Citado na pág. 7

Carbone *et al.* (2005) Pedro Paulo Carbone, Hugo Pena Brandão e João Batista Diniz Leite. *Gestão por competências e gestão do conhecimento*. FGV. Citado na pág. 7, 8

Chelimsky *et al.* (2010) David Chelimsky, Dave Astels, Bryan Helmkamp, Dan North, Zach Dennis e Aslak Hellesoy. *The RSpec book: Behaviour driven development with Rspec, Cucumber, and friends*. Pragmatic Bookshelf. Citado na pág. 15

e Silva e Michaelis (1998) Adalberto Prado e Silva e Henriette Michaelis. *Michaelis: moderno dicionário da língua portuguesa*. Melhoramentos. Citado na pág. 3

Fowler (2002) Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc. Citado na pág. 12

Hansson (2014a) David Heinemeier Hansson. Rails, web development that doesn't hurt, 2014a. URL http://rubyonrails.org/. Acesso em 15/10/2014. Citado na pág. 18

Hansson (2014b) David Heinemeier Hansson. Ruby on rails guides, 2014b. URL http://guides.rubyonrails.org/. Acesso em 15/10/2014. Citado na pág. 18

Hansson (2014c) David Heinemeier Hansson. A guide to testing rails applications, 2014c. URL http://guides.rubyonrails.org/testing.html. Acesso em 15/10/2014. Citado na pág. 15

Hellesøy *et al.* (2014) Aslak Hellesøy, Joseph Wilk, Matt Wynne, Gregory Hnatiuk e Mike Sassa. Cucumber: Behaviour driven development with elegance and joy, 2014. URL http://cukes.info/. Acesso em 15/10/2014. Citado na pág. 15

Hipp (2014) D Richard Hipp. Sqlite, 2014. URL http://www.sqlite.org. Acesso em 15/10/2014. Citado na pág. 6

Kim (1990) Won Kim. *Introduction to object-oriented databases*, volume 90. MIT press Cambridge, MA. Citado na pág. 12

Martin (2008) Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education. Citado na pág. 18

Matsumoto (2014) Yukihiro Matsumoto. Ruby, o melhor amigo do programador, 2014. URL https://www.ruby-lang.org/pt/. Acesso em 15/10/2014. Citado na pág. 18

Olszowka (2014) Christoph Olszowka. Simplecov, 2014. URL https://github.com/colszowka/simplecov. Acesso em 15/10/2014. Citado na pág. 16

Oracle (2014) Oracle. Core j2ee patterns - data access object, 2014. URL http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. Acesso em 15/10/2014. Citado na pág. 12

PYPL (2014) PYPL. Pypl popularity of programming language index, 2014. URL https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language. Acesso em 15/10/2014. Citado na pág. 9

Setzer (1999) Valdemar W Setzer. Dado, informação, conhecimento e competência. *DataGrama-Zero Revista de Ciência da Informação, n. 0. http:// www.ime.usp.br/ ~vwsetzer/ dado-info.html*. Citado na pág. 1, 2, 3, 6, 7

Setzer (2000) Valdemar W Setzer. Exemplos de matrizes do projeto da prodesp (comunicação pessoal), 2000. Citado na pág. 1

Setzer e da Silva (2005) Valdemar Waingort Setzer e Flávio Soares Corrêa da Silva. *Bancos de dados: aprenda o que são, melhore seu conhecimento, construa os seus*. Edgard Blücher. Citado na pág. 11

Sommerville (2004) Ian Sommerville. *Software Engineering. International computer science series*. Addison Wesley. Citado na pág. 12, 15

Taylor *et al.* **(2009)** Richard N Taylor, Nenad Medvidovic e Eric M Dashofy. *Software architecture: foundations, theory, and practice*. Wiley Publishing. Citado na pág. 11

TIOBE (2014) TIOBE. Tiobe index for october 2014, 2014. URL http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Acesso em 15/10/2014. Citado na pág. 9

Weilkiens e Oestereich (2010) Tim Weilkiens e Bernd Oestereich. *UML 2 Certification Guide: Fundamental & Intermediate Exams*. Morgan Kaufmann. Citado na pág. 6