UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

Sistema de concessão de financiamento para a apresentação de artigos acadêmicos em Spring MVC

Rafael Fogel

Orientador
Márcio de Oliveira Barros

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2011

Sistema de concessão de financiamento para a apresentação de artigos acadêmicos em Spring MVC

Rafael Fogel

Aprovado em ____/_____/_____

BANCA EXAMINADORA

_____

Prof. Márcio de Oliveira Barros, DSc. (UNIRIO)

_____

Prof. Leila Cristina Vasconcelos de Andrade, DSc. (UNIRIO)

_____

Prof. Vânia Maria Félix Dias, DSc. (UNIRIO)

Rio de Janeiro, ____ de _____ de _____

Rafael Fogel

Agradecimentos

Ao professor Márcio Barros pela orientação, empenho e paciência durante todo o desenvolvimento do projeto.

Às professoras Vânia Maria Félix e Leila Andrade por terem aceitado o convite para fazer parte da banca. Esta atitude reafirmou todo o respeito que tenho por ambas.

Aos amigos e familiares que apoiaram esta empreitada, sempre apoiando nas horas difíceis e dando força para que o projeto fosse concluído no tempo.

RESUMO

Este projeto visa apresentar o framework para desenvolvimento de aplicações de servidor Web, Spring MVC, um framework de código aberto para desenvolvimento de aplicações Web em Java. O Spring MVC é baseado nos padrões de projeto de inversão de controle (IoC) e injeção de dependência. Ele possui uma arquitetura baseada em interfaces e POJO's (Plain Old Java Objects), oferecendo, através deles, características como mecanismos de segurança e controle de transações, simplificando o desenvolvimento.

Para demonstrar a implementação do Spring MVC, foi desenvolvido um sistema para apoio ao processo de concessão de financiamento para a apresentação de publicações acadêmicas do DIA (Departamento de Informática Aplicada). Atualmente, o controle da concessão de financiamento para as publicações é feito através de planilhas e arquivos de texto simples, carecendo de sistema mais eficaz, ágil e seguro. Sendo assim, foi criado um portal na Internet para reunir essas publicações em um só lugar, facilitando o trabalho dos professores. O sistema funciona via Web, facilitando seu uso e manutenção. Nele, é possível gerenciar pedidos que serão avaliados pela Comissão de Mérito.

Palavras-chave: Contrução de software, Spring MVC, Portal Web, Internet, Java

ABSTRACT

This project aims to introduce Spring MVC, an open-source framework for Java Web application development. Spring MVC is based on inversion of control and dependency injection design patterns. It relies on an architecture based on interfaces and POJO's, offering features like security mechanisms and transaction controllers, which simplify the development.

To demonstrate an implementation using Spring MVC, we have developed a system to support the process of funding the presentation of academic papers by teachers and researchers composing the DIA (Applied Computing Department). Currently, the process of funding these papers is controlled by Excel spreadsheets and simple text files, lacking of a more efficient, safe, and fast system. Thus, it was created a Web portal to gather this information in a single place, facilitating the teacher's work. The system resides on the Web, facilitating its usage and maintenance. It allows managing the paper presentation requests that will be evaluated by a Merit Commission, following pre-established rules and workflows.

Keywords: Software Development, Spring MVC, Web Portal, Internet, Java

Índice

Índice de Figuras

Índice de Códigos

1 Introdução

1.1 Motivação

A motivação desse projeto vem de uma carência da Comissão de Mérito para avaliação de artigos acadêmicos do Departamento de Informática Aplicada da UNIRIO. Atualmente, esta comissão controla a concessão de financiamento das publicações através de planilhas Excel e arquivos simples de texto, dificultando o seu controle, atrasando e comprometendo a integridade do processo de avaliação de artigos. A criação de um sistema Web trará agilidade e dinamismo ao processo, facilitando a comunicação entre professores.

Outro fator motivacional foi trabalhar com o Spring MVC [SPRINGSOURCE, 2011], um framework de desenvolvimento de software para servidor Web em ascensão no mercado de trabalho. O Spring é considerado um dos frameworks mais completos em Java, recomendado para qualquer projeto, independente de seu tamanho [SPRINGSOURCE, 2011]. Sua versatilidade impressiona com o poderoso sistema de anotações, que simplifica e reduz a codificação e a configuração da aplicação. Sua estrutura foi implementada de forma que o desenvolvedor não tenha que perder muito tempo com arquivos XML, aumentando sua produtividade.

1.2 Objetivos

O objetivo deste projeto foi criar um sistema Web capaz de gerenciar e automatizar o processo de concessão de financiamento para a apresentação de publicações acadêmicas do DIA. Para tal, foi utilizado o framework Spring MVC, um framework de código aberto em plataforma Java.

1.3 Organização do texto

O presente trabalho está estruturado em capítulos e será desenvolvido da seguinte forma:

- Capítulo II: Desenvolvimento Web Com Spring - apresenta ao leitor as tecnologias que foram usadas neste projeto, principalmente o Spring MVC, suas camadas e algumas de suas funcionalidades. Também é feita uma análise sobre as vantagens e desvantagens sobre o uso de anotações;

- Capítulo III: O Sistema de Avaliação de Mérito do PPGI/UNIRIO - fala sobre o Sistema de Avaliação de Mérito, produto deste projeto, focando nos detalhes e implementação, como as suas classes, a arquitetura do sistema, e a estrutura de seu banco de dados;

- Capítulo IV: A Implementação do Sistema de Avaliação de Mérito do PPGI/UNIRIO – introduz ao leitor as funcionalidades do Sistema de Avaliação de Mérito, focando nos três tipos de usuários do sistema: o professor, o Membro da Comissão de Mérito e o Presidente da Comissão de Mérito. Também é feita uma discussão sobre o uso do Spring MVC neste projeto, reconhecendo suas vantagens e desvantagens.

- Capítulo V: Conclusões – Reúne as considerações finais, assinala as contribuições da pesquisa e sugere possibilidades de aprofundamento posterior.

2 Desenvolvimento Web com Spring

Este capítulo trata do framework Spring MVC, mostrando algumas de suas funcionalidades e apresentando sua relação com o modelo MVC (Modelo – Visão – Controle). Também são apresentadas as anotações do Spring, consideradas uma das grandes vantagens de sua utilização.

O capítulo começa falando sobre frameworks, explicando seu conceito e citando algumas de suas características. São apresentados também alguns frameworks conhecidos no mundo dos desenvolvedores para a Web, incluindo o Spring MVC. Em seguida, o capítulo aborda o conceito de MVC, explicando suas camadas e fazendo uma análise das mesmas no contexto do Spring MVC. Por fim, o capítulo faz uma apresentação detalhada do Spring MVC, explicando sua estrutura, principais funcionalidades e camadas.

2.1 Frameworks

O conceito de framework é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica, ou seja, ele captura funcionalidades comuns a várias aplicações e provê uma solução. Um framework não se trata de um software executável, mas sim de uma estrutura de classes às quais o desenvolvedor tem que se adaptar, com o objetivo de auxiliar o desenvolvimento de software. Os frameworks possuem muitas vantagens tais como [ZEMEL(a), 2011]:

- **Utilidade:** os frameworks possuem funcionalidades das mais variadas, que auxiliam a resolver questões de programação do dia-a-dia com qualidade e eficiência;
- **Segurança:** frameworks são projetados de modo a garantir a segurança da aplicação, pois eles são desenvolvidos utilizando as implementações de segurança mais utilizadas;
- **Capacidade de Extensão:** os frameworks permitem estender suas funcionalidades nativas. Se uma determinada biblioteca não contempla todas as funcionalidades desejadas, ela permite que o usuário estenda suas funcionalidades e as use como se fossem parte do framework;
- **Economia de tempo:** trechos de códigos repetitivos, que levariam dias para se fazer, são disponibilizados automaticamente pelos frameworks;
- **Suporte:** os criadores de frameworks geralmente trabalham com uma comunidade grande de desenvolvedores que auxiliam o projeto, seja com código ou testes, e disponibilizam material de qualidade nos sites ou repositórios oficiais, formando uma vasta documentação a respeito. Além disso, eles estão dispostos a se ajudar entre si.

2.2 Arquitetura MVC (Model – View – Control)

A arquitetura MVC (Model – View – Controller) pode ser entendida como uma divisão de tarefas em um sistema de software. Com o desenvolvimento e evolução dos softwares e, consequentemente, da forma de se fazer os softwares, novas abordagens tiveram que ser pensadas para facilitar a programação e garantir que, depois de prontos, fossem mais manuteníveis. A partir disso, surgiu o conceito de dividir tarefas, de garantir que cada "camada" da aplicação tenha seu próprio escopo e definição e que a comunicação entre elas seja feita de maneira eficiente e controlada.

A figura 1 apresenta a relação entre as camadas da arquitetura MVC. Estas camadas podem ser definidas da seguinte forma [ZEMEL(b), 2011]:

- **Model:** em um software baseado em MVC, é o modelo que representa as informações do domínio do problema. O modelo é usado para definir e gerenciar o domínio da informação e notificar observadores sobre mudanças nos dados. Ele é uma representação detalhada da informação que a aplicação opera;
- **View:** é a apresentação, é o que aparece, o que é visualizado por quem usa o sistema. É na camada de visão que as informações, sejam elas quais forem e de qual lugar tenham vindo, são exibidas para o usuário;
- **Controller.** como sugere o nome, é responsável por controlar todo o fluxo do programa. É o "cérebro" e o "coração" do software. É na camada de controle que se decide "se", "o que", "quando", "onde" e tudo o mais que faz com que a lógica da aplicação funcione. Desde o que deve ser consultado no banco de dados à tela que vai ser exibida, é no Controller que isso deve ser definido.

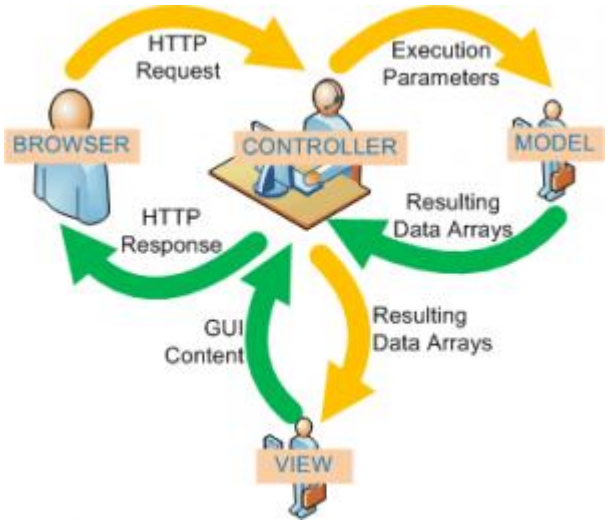

Figura 1 – Visão geral da arquitetura MVC em aplicação Web

O modelo MVC possui algumas vantagens tais como [MACORATTI, 2011]:

- **Facilidade do uso:** como o modelo MVC gerencia múltiplos visualizadores usando o mesmo modelo, é fácil manter , testar e atualizar múltiplos sistemas;
- **Simples de se usar:** é simples incluir novos clientes na aplicação, apenas incluindo seus visualizadores e controles;
- **Independência**: é possível ter desenvolvimento em paralelo para o modelo, visualizador e controle, pois são elementos independentes do sistema.

2.3 Frameworks de Desenvolvimento para Servidor Web

Os frameworks de desenvolvimento de software para servidor Web são considerados peças fundamentais no desenvolvimento de software e alguns deles se destacam entre os demais. Um deles é o Struts, um framework para Web que se tornou padrão na comunidade JEE (Java Enterprise Edition). Ele é um framework de desenvolvimento da camada controladora, em uma estrutura que segue o padrão Model 2 (MVC + Front Controller). No Struts, o *Front Controller* é o componente responsável pelo processamento das requisições da aplicação, centralizando funções como seleção de visão, segurança, etc. O Struts foi originalmente desenvolvido por Craig McClanahan e doado para a Apache Software Foundation em 2002, onde continua sendo desenvolvido segundo os padrões desta fundação [STRUTS, 2011].

No Struts, os componentes do controlador são responsáveis pelo fluxo da aplicação. O principal componente controlador do Struts é o *ActionServet*, que é uma extensão de Servlet. Sua principal função é fazer o mapeamento das requisições do servidor. Para isso, é necessário criar um arquivo de configuração, denominado *struts-config.xml*, que é usado para mapear a navegação da aplicação. Depois disso, devem ser criadas as classes *Action* (que são extensões da classe *Action* do Struts), que conterão as

ações a serem executadas a cada requisição. O objetivo da classe *Action* é processar a requisição e retornar um objeto da classe *ActionForward*, que identifica para qual componente de visão (normalmente uma página JSP) será gerada a resposta [EDUARDO, 2011].

Outro framework bastante conhecido é o Java Server Faces, ou JSF [JAVAFACES, 2011]. Mais que um framework para desenvolver aplicações web de forma ágil, JSF foi incorporado à especificação J2EE. Ele é atualmente considerado pela comunidade Java como a última palavra em termos de desenvolvimento de aplicações Web utilizando Java, resultado da experiência e maturidade adquiridas com o JSP/Servlet (Model1), Model2 (MVC) e Struts [FACES, 2011].

O JSF possui algumas vantagens, como os controles GUI personalizados, onde ele fornece um conjunto de API's para criar formulários HTML que possuem interfaces complexas e a facilidade para designar o código Java que é chamado quando os formulários são submetidos. O código pode responder a botões particulares, mudanças em valores específicos, seleções de determinado usuário, e assim por diante [HALL, 2005]. As configurações do JSF ficam no arquivo *faces-config.xml*, que é responsável por descrever os elementos que compõem o projeto, tais como as regras de navegação, os objetos gerenciados, configurações de localização, entre outros

2.4 Spring MVC

O Spring MVC é um framework de código aberto para a plataforma Java criado por Rod Johnson [JOHNSON, 2002]. Trata-se de um framework não intrusivo, baseado nos padrões de projeto de inversão de controle (IoC) e injeção de dependência. Ele possui algumas vantagens, tais como [SPRINGSOURCE, 2011]:

- Flexibilidade**,** pois ele é totalmente baseado em interfaces. Assim, cada pedaço do framework é configurável através de sua própria interface;
- A camada de controle pode ser configurada através da IoC (Inversão de controle), como qualquer outro objeto. Isso facilita a execução de testes além de criar uma integração robusta com outros objetos manipulados pelo Spring MVC;
- Diferente do Struts, que obriga o desenvolvedor a estender certas classes do framework na camada de controle, o Spring MVC, quando usado com as anotações, dá liberdade ao desenvolvedor para que ele programe da maneira que desejar, embora seja conveniente a utilização das classes do framework;
- O Spring MVC suporta diversas tecnologias na camada de visão, como o *Velocity* e *XLST*, além do *JSP*.

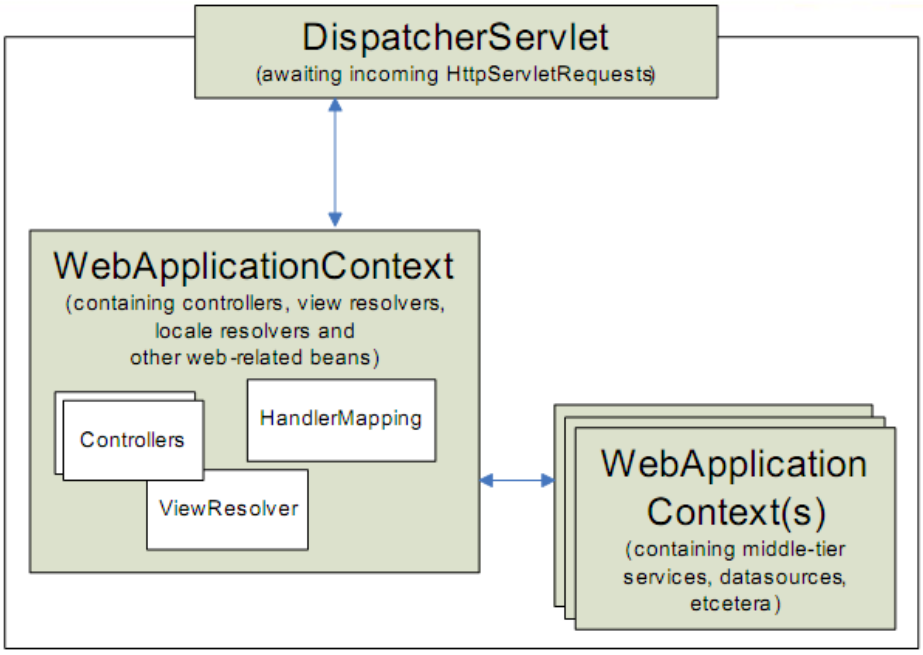

Figura 2 – Visão geral da arquitetura do Spring MVC

O *DispatcherServlet*, referenciado na figura 2, é o responsável por receber todas as requisições do cliente, executando a parte comum destes pedidos e delegando implementações específicas para os controladores [JAVABEAT, 2011].

Um controlador representa o componente que recebe o *HttpServletRequest* e o *HttpServletResponse*, servindo como um servlet. Porém, é capaz de participar do workflow do MVC. Para que o Spring reconheça um controlador, é necessário registrá-lo no arquivo de configuração de servlets (*dispatcher-servlet*), conforme no exemplo do código 1.

```
<bean id="alterarSenhaFormController" class="Controller.AlterarSenhaFormController">
    <property name="validator" ref="AlterarSenhaFormValidator" />
    <property name="formView" value="AlterarSenhaForm.jsp" />
    <property name="successView" value="TemplateForm.html" />
    <property name="commandName" value="loginData" />
    <property name="commandClass" value="POJO.Login" />
</bean>
```

Código 1 - Registro do controlador

O controlador possui algumas propriedades que podem ser definidas no *dispatcher-servlet*, tais como:

- **validator:** classe de validação de dados submetidos pelo usuário;
- **formView:** é a página que será carregada quando o controlador for invocado;
- **sucessView:** é a página que será carregada quando um usuário executar um *post* na página com sucesso.

Também é possível definir uma espécie de objeto da *view*, que será relacionado a uma classe POJO do sistema. Isso é útil quando se tem mais de um formulário na página, pois não é necessário capturar as informações individualmente nos *requests*, ou quando se deseja apresentar as informações em um formulário. O Spring encapsula

essas informações em um objeto (*commandName*) do tipo referenciado (*commandClass*) facilitando seu controle e validação**.**

O *HandlerMapping* e o *viewResolver* são interfaces do Spring MVC implementadas por objetos que servem para definir o mapeamento das páginas e seus respectivos controladores.

O workflow do Spring MVC é definido da seguinte forma: quando o *DispatcherServlet* recebe um pedido (*request*), ele tenta localizar o controlador responsável pela requisição através do *HandlerMapping*. Quando um controlador é achado, o método *get* do controlador é invocado, o qual é responsável pelo tratamento do pedido e, se for o caso, retorna um *ModelAndView* apropriado. O *ModelAndView* é uma classe do Spring, que representa o modelo e a visão que devem ser utilizados. Conforme informado na documentação da API, essa classe "representa um modelo e uma visualização retornados por um manipulador, a serem resolvidos por um DispatcherServlet." [SPRINGSOURCE, 2011]. Sendo assim, o *ModelAndView* se encarrega de invocar a página requisitada. Se nenhuma página for especificada no método *get*, a página definida na propriedade *formView* (Código 1) é carregada.

2.5 A Camada de Controle (Controlador)

A camada de controle é a interface de comunicação entre as páginas Web e o modelo de negócio. Como foi dito anteriormente, o controlador determina o fluxo execução do sistema, servindo como uma camada intermediária entre a camada de apresentação e o modelo, gerando os códigos dinâmicos em HTML.

Qualquer implementação de uma interface de controlador deve ser reutilizável, segura e capaz de processar múltiplas requisições durante o ciclo de vida da aplicação. Para se criar um controlador, é necessário estender ou implementar alguma classe do

Spring, ficando a cargo do usuário selecionar a mais indicada. O Spring MVC possui algumas classes e interfaces específicas para se trabalhar com controlador, porém duas delas se destacam: *Controller* e *SimpleFormController*. *Controller* é uma interface indicada para páginas comuns, sem envio de dados por parte do usuário. *SimpleFormController*, que estende a classe *AbstractController*, é a classe indicada para páginas com formulários, onde o usuário entrará com informações que serão interpretadas pela aplicação.

```
public class AlterarSenhaFormController extends SimpleFormController {
	@Override
	public ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response,
	Object command, BindException errors) throws Exception
	{
		Login login = (Login)command;
		int id_login = (Integer)request.getSession().getAttribute("id_login");
		login.setId(id_login);
		LoginBusiness logBus =
		BusinessFactory.Instance(BusinessFactoryImpl.class).getLoginBusiness();
		try{
		logBus.editarLogin(login);
		}catch (BusinessException e) {
		e.printStackTrace();
		}
		ModelAndView modelAndView = new ModelAndView(new
		RedirectView(getSuccessView()));
		return modelAndView;
	}
```

Código 2 - Exemplo de controlador e seu método *post*

O código 2 mostra um exemplo de controlador estendendo a classe *SimpleFormController*. O método *onSubmit* é sobrecarregado e é invocado no *post*. Ele retorna um *ModelAndView* da página especificada na propriedade *successView* do controlador.

```
@Override
protected Object formBackingObject(HttpServletRequest request)throws Exception
{
    LoginBusiness logBus =
    BusinessFactory.Instance(BusinessFactoryImpl.class).getLoginBusiness();
    Login log = null;
    try {
         int id_login = (Integer)request.getSession().getAttribute("id_login");
         log = logBus.consultarLogin(id_login);
       } catch (BusinessException e)
       {
         e.printStackTrace();
       }
    log.setPassword("");
    return log;
}
```

Código 3 - Exemplo de controlador e seu método *get*

O código 3 mostra o método *formBackingObject* do mesmo controlador apresentado no código 2. Esse método é invocado no *get*. Ele retorna um objeto que será atribuído ao objeto de sessão especificado na propriedade *commandName* (Código 1) do controlador.

```
public class LogoutFormController implements Controller {
            @Override
            public ModelAndView handleRequest(HttpServletRequest hsr, HttpServletResponse hsr1)
            throws Exception {
                HttpSession session = hsr.getSession();
                session.invalidate();
            ModelAndView modelAndView = new ModelAndView(new     RedirectView("Login.html"));
                return modelAndView;
           }
}
```

Código 4 – Outro exemplo de controlador e seu método *get*

O código 4 apresenta um controlador implementando a interface *Controller*. Essa interface possui apenas um método, o *handleRequest*. Ele retorna um *ModelAndView* de uma página que não foi especificada nas propriedades do controlador. Portanto, ela precisará estar registrada nas interfaces de mapeamento para

ser invocada. Para se registrar uma página e atribuí-la ao seu controlador, voltamos ao *dispatcher-servlet* e adicionamos as seguintes linhas, mostradas no código 5.

```
<bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
   <property name="urlMap">
         <map>
               <entry key="Login.html">
                  <ref bean="loginFormController" />
               </entry>
     </map>
   </property>
</bean>
```

Código 5 – Registro de uma view com seu respectivo controlador

Desta forma, toda vez que a página "Login.html" for requisitada, o *dispatcher-servlet* saberá que o controlador responsável por essa página é o "loginFormController". O processo de registrar cada controlador com sua respectiva *view* tende a ser dispendioso e cansativo. Para acabar com esse problema, desde a versão 2.5, o Spring MVC conta com sistema de anotações, assunto que será abordado mais a frente neste capítulo.

O *ModelAndView* ainda permite a inclusão de parâmetros em seu construtor para que possa ser atribuído à URL, dispensando o usuário de ter que fazer isso manualmente, como pode ser visto no código 6.

```
public class RelatorioFormController extends AbstractController {
        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest request,
        HttpServletResponse response) throws Exception{
            Pedido pedido = (Pedido)request.getAttribute("pedido");
            Map<String, Object> params = new LinkedHashMap<String, Object>();
            params.put("autor", pedido.getAutor());
            params.put("qualificacao", pedido.getQualificacaoConferencia());
            params.put("nomeConferencia", pedido.getNomeConferencia());
            params.put("nomeArtigo", pedido.getNomeArtigo());
            params.put("financiamento", pedido.getFinanciamento());
            params.put("cota", pedido.getCota());
            params.put("numeroArtigos", pedido.getNumeroArtigos());
            Date data = new Date();
            SimpleDateFormat formatador = new SimpleDateFormat("dd/MM/yyyy");
            String dataFormatada = formatador.format(data);
            params.put("dataHora", dataFormatada);

             int pontos = CalculaPontos(pedido);
             String conceito = CalculaConceito(pontos);
      int valorSugerido = CalculaFinanciamentoSugerido(conceito,   pedido.getCota());
     String moeda = ( pedido.getCota().equals("Cota Internacional") ? "U$" : "R$" );

      params.put("conceito", conceito);
                params.put("valorSugerido", valorSugerido);
                params.put("moeda", moeda);

                ModelAndView modelAndView = new ModelAndView("RelatorioForm.jsp", "params",
                params);
      return modelAndView;
      }
```

Código 6 – Exemplo da utilização do *ModelAndView* com parâmetros

2.6 A Camada de Visão (View)

A camada de visão é um intermediário entre o programa e o usuário. Ela é responsável por construir uma interface para interação, preparando a apresentação da informação para que o usuário possa reagir às respostas e continuar utilizando o programa. Em Java, utilizaremos páginas *JSP* com *JSTL* para produzir as páginas de resposta. A camada de visão não acessa diretamente o banco de dados nem realiza nenhuma lógica de negócio: ela recebe objetos gerados pelas classes da camada de controle e produz código HMTL a partir deles.

Figura 3 - Exemplo de View

A figura 3 mostra um exemplo básico de uma visão com 2 campos de entrada de dados e um botão para envio de requisição ao servidor.

```
<%@page contentType="text/html"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

<html>
  . . .
      <form:form commandName="loginData" method="POST">
       <table>
       <tr>
         <td width="55">Usuário: </td>
         <td><form:input path="username"/></td>
         <td><form:errors path="username" cssClass="error" /></td>
       </tr>
       <tr>
         <td width="55">Senha: </td>
         <td><form:input type="password" path="password"/></td>
         <td><form:errors path="password" cssClass="error" /></td>
       </tr>
       </table><br>
       <center><input type="image" src="Imagens/entrar.png" value="submit"/></center>
       </form:form>
  . . .
</html>
```

Código 7 - Código da View da figura 3

O código 7 utiliza uma notação exclusiva do Spring MVC, o *form*. Essa notação é derivada da tag de mesmo nome do *HTML* e também serve para criar um formulário na página, porém com uma função adicional. O Spring MVC reconhece essa notação e consegue encapsular seus campos em um objeto de uma classe do sistema (*commandClass*). Conforme mostrado no *Código 1*, o usuário pode registrar o nome desse objeto na propriedade *commandName* e referenciá-lo na *view*. Desta forma, é possível tanto enviar objetos para *view* quanto recuperá-los.

2.7 Validação de Dados

A camada de validação de dados é responsável por validar as informações recebidas do usuário. Nela, as informações passam por um controle de validação que verifica se as informações estão de acordo com as regras de negócio pré-estabelecidas.

Toda vez que o usuário executa uma ação no sistema onde há entrada de dados, a camada de controle verifica se essa ação necessita passar por uma validação antes de ser consumida pelo sistema. Caso necessite, a camada de validação é chamada e inicia-se o processo de verificação dos dados. Esse processo analisa cada informação passada pelo usuário e testa sua validade. Caso a informação seja rejeitada, uma notificação de erro é reportada e a camada de controle interrompe a operação.

```
public class AlterarSenhaFormValidator implements Validator {
  @Override
  public void validate(Object obj, Errors errors)
  {
    Login login = (Login)obj;

    if (login.getUsername() == null || login.getUsername().length() == 0)
      errors.rejectValue("username", "error.empty.field", "(Obrigatório)");

    if (login.getPassword() == null || login.getPassword().length() == 0)
      errors.rejectValue("password", "error.empty.field", "(Obrigatório)");

    if (login.getNovoPassword() == null || login.getNovoPassword().length() == 0)
      errors.rejectValue("novoPassword", "error.empty.field", "(Obrigatório)");

    else
    {
      if (!login.getNovoPassword().equals(login.getPassword()))
        errors.rejectValue("novoPassword", "error.empty.field", "(As senhas não são      iguais!)");
    }

  }

  @Override
  public boolean supports(Class clazz)
  {
    return clazz.equals(Login.class);
  }
}
```

Código 8 - Exemplo de classe de validação

No Spring MVC, para se criar uma classe de validação é necessário estender a classe *Validator*. O método *validate* recebe o objeto da visão (*commandName* – Código 1) vindo do controlador com as informações passadas pelo usuário e o relaciona a uma classe do sistema. Esse relacionamento é feito através do método *support.* No exemplo do código 8, o objeto da visão será atribuído à classe *Login*, permitindo a sua manipulação pelo usuário.

2.8 Anotações de Código no Spring

Anotação é um mecanismo que utiliza metadados no código para relacioná-lo a alguma ação pré-definida. As anotações surgiram como uma forma de minimizar os arquivos XML de configuração, que tornam muito difícil a compreensão de alguns sistemas e tomam muito tempo de desenvolvimento.

O uso de anotação é recomendado pela sua facilidade e simplificação de código. Com ele, registrar os controladores e mapeá-los com suas respectivas *views* no arquivo de configuração não é mais necessário. Porém, é preciso indicar ao Spring o uso das anotações direto nos controladores. Para isso, é preciso adicionar as bibliotecas do Spring no arquivo de configuração de servlets, o *dispatcher-servlet*.

```
<context:component-scan base-package="Controller" />
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />
```

Código 9 – Registro das anotações no *dispatcher-servlet*

O código 9 apresenta a classe necessária para se utilizar as anotações, a *AnnotationMethodHandlerAdapter*. Também é necessário indicar a localização dos controladores da aplicação, conforme feito na primeira linha do código 9.

```
@Controller
@RequestMapping("AlterarSenha.html")
@SessionAttributes("login")
public class AlterarSenhaFormController {
  private AlterarSenhaFormValidator alterarSenhaFormValidator;
  private LoginBusiness logBus;

  @Autowired
  public AlterarSenhaFormController(AlterarSenhaFormValidator alterarSenhaFormValidator)
  {
    this.alterarSenhaFormValidator = alterarSenhaFormValidator;
    this.logBus =      BusinessFactory.Instance(BusinessFactoryImpl.class).getLoginBusiness();
  }
```

Código 10 – Exemplo de controlador utilizando anotações

As anotações são identificadas pelo uso do arroba (@). No código 10 foram usadas quatro anotações:

- **@Controller:** indica que essa classe faz parte da camada de controle. Esta anotação permite que o Spring faça uma varredura automática nas classes através do elemento <*context:component-scan*>, indicado no código 9;
- **@RequestMapping:** indica qual view esse controlador irá retornar quando lhe for requisitado;
- **@SessionAttributes:** se refere ao objeto da view (*commandName* – Código 1) que será criado para esse controlador. Como foi explicado anteriormente, o Spring MVC consegue encapsular informações de entrada enviadas pelo usuário em um único objeto, facilitando sua manipulação e validação;
- **@Autowired:** é específica para construtores. Ela aplica o conceito de Inversion of Control (IoC), ou Inversão de Controle. Este é o nome dado ao padrão de desenvolvimento de software onde a seqüência (controle) de chamadas dos métodos é invertida em relação à programação tradicional, ou seja, ela não é determinada diretamente pelo programador [FOWLER, 2011]. Nesse caso, a aplicação instancia o objeto passado como parâmetro e atribui à variável da classe. Os campos são preenchidos após as construções das classes e antes de qualquer método ser chamado.

O código 11 apresenta a mesma classe e método do código 2, porém com a utilização de anotação. A anotação *@RequestMapping* determina que o método *onSubmit* executa a função de *post*. Sendo assim, é possível utilizar qualquer assinatura de método, excluindo e adicionando parâmetros que o usuário julgue ser necessário.

```
@RequestMapping(method = RequestMethod.POST)
  public String onSubmit(@ModelAttribute("login") Login login, BindingResult result,
          HttpServletRequest request) throws Exception{
    alterarSenhaFormValidator.validate(login, result);
    if (result.hasErrors())
      return "AlterarSenhaForm.jsp";
    int id_login = (Integer)request.getSession().getAttribute("id_login");
    login.setId(id_login);
    try
    {
      logBus.editarLogin(login);
    }
    catch (BusinessException e)
    {
      e.printStackTrace();
    }
    return "redirect:TemplateForm.html";
  }
```

Código 11 - Exemplo de controlador utilizando anotações e seu método *post*

Quando utilizamos controladores derivados das classes do Spring MVC, a validação, quando especificada, é chamada automaticamente. Usando anotações, devemos chamá-la manualmente com a função *validate*, conforme o código 11.

```
@RequestMapping(method = RequestMethod.GET)
  public String showUserForm (ModelMap model, HttpServletRequest request)
  {
    Login log = null;
    try
    {
      int id_login = (Integer)request.getSession().getAttribute("id_login");
      log = logBus.consultarLogin(id_login);
    }
    catch (BusinessException e)
    {
      e.printStackTrace();
    }

    log.setPassword("");

    model.addAttribute(log);
    return "AlterarSenhaForm.jsp";
  }
```

Código 12 - Exemplo de controlador utilizando anotações e seu método *get*

Utilizando anotações, não é mais necessário utilizar a classe *ModelAndView* para invocar outras páginas. Os métodos *post* e *get* dos códigos 11 e 12, respectivamente, retornam strings. Essas strings são interpretadas pelo *viewResolver*, exatamente igual aos métodos do *ModelAndView*.

Para concluir o assunto sobre anotações, ficou claro que seu uso aumenta a produtividade e facilita a vida do programador, pois reduz a configuração via XML, além da flexibilidade de se poder criar métodos personalizados para os controladores. Em contrapartida, usar somente XML permite ao usuário concentrar todas as configurações pertinentes às camadas do modelo MVC em um só lugar, facilitando seu controle.

2.9 Considerações Finais

Neste capítulo, foram abordadas algumas características do Spring. Foi feita uma análise individual sobre as camadas do framework (modelo, visão e controle), além da camada de validação, todos com exemplo prático do sistema desenvolvido. Também foi falado sobre o sistema de anotações do Spring, considerado uma de suas maiores vantagens.

O próximo capítulo falará sobre o Sistema de Avaliação de Mérito do DIA/Unirio. Esse sistema, implementado em Spring, foi uma das motivações desse projeto. O capítulo explicará passo a passo o workflow de avaliação dos pedidos, como funciona o sistema em modo geral e apresentará detalhes de sua implementação.

3 O Sistema de Avaliação de Mérito do DIA/UNIRIO

Este capítulo tratará do sistema Web desenvolvido em Spring MVC para a gestão do processo de avaliação de mérito para concessão de financiamento para artigos acadêmicos pela Comissão de Avaliação de Mérito do DIA/UNIRIO, que é o produto final deste projeto. Serão apresentados a arquitetura do sistema, as classes implementadas, o modelo do banco de dados utilizado, os casos de uso do sistema e seus respectivos diagramas.

3.1 Os Processos de Avaliação de Mérito

A Comissão de Avaliação de Mérito tem o objetivo de apoiar a participação de professores do Departamento de Informática Aplicada e dos alunos de pós-graduação em eventos nacionais e internacionais com fins de apresentação de trabalhos científicos. Para organizar as avaliações dos artigos científicos submetidos à Comissão, foi criado um Processo de Avaliação de Mérito definido em quatro etapas e que é regido por uma comissão formada por um presidente e dois membros.

Segundo este processo, a avaliação é disparada quando o professor envia uma proposta com dados preenchidos sobre o seu artigo. Os dados necessários incluem a cota de participação em eventos que será utilizada, o nome do artigo, o nome da conferência, a classificação Qualis da conferência, uma indicação de pedido de

financiamento externo (às agências de fomento) e o número de artigos aceitos na conferência ou seus eventos satélite. Essas informações são necessárias para que a Comissão de Mérito calcule o valor do financiamento sugerido, segundo as regras estabelecidas para esta Comissão e aprovadas no Colegiado do Departamento. Cada professor tem direito a três cotas anuais de participação em eventos científicos, sendo duas nacionais e uma internacional.

Na segunda etapa, o presidente da Comissão de Mérito avalia o pedido e verifica se as informações estão corretas e coerentes, aceitando ou rejeitando o pedido de acordo com a situação. Na terceira etapa, os membros da Comissão de Mérito avaliam o pedido em relação ao custo e benefício, levando em conta a sua classificação. Os dois membros devem chegar a um acordo, seja aceitando ou rejeitando o pedido de avaliação de mérito. Na quarta e última etapa, o presidente homologa a decisão dos membros. A figura 4 mostra um diagrama de estado para os pedidos de avaliação de mérito.

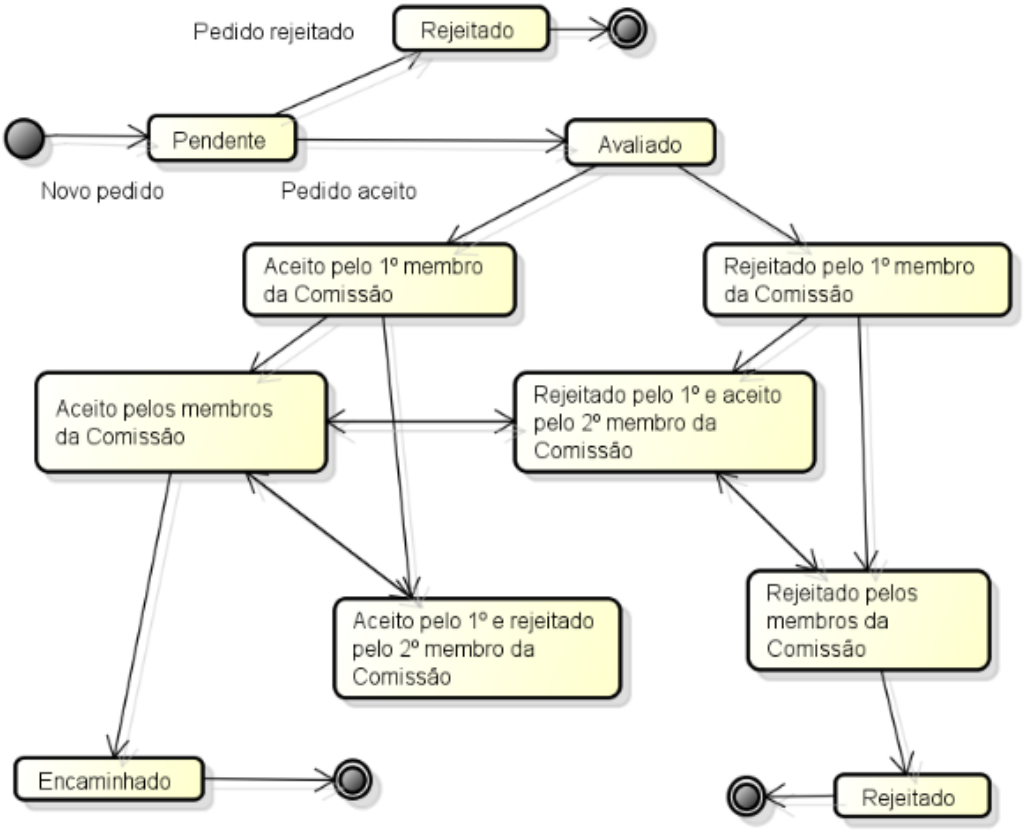

Figura 4 – Diagrama de estado dos pedidos

Os pedidos são classificados seguindo critérios pré-definidos pela Comissão de Mérito. Para começar, os pedidos recebem uma pontuação pela qualificação do evento no sistema QUALIS de Computação, segundo a seguinte regra:

- Estrato QUALIS A1: 20 pontos;
- Estrato QUALIS A2: 18 pontos;
- Estrato QUALIS B1: 16 pontos;
- Estrato QUALIS B2: 12 pontos;
- Estrato QUALIS B3: 10 pontos;
- Estrato QUALIS B4: 8 pontos;
- Estrato QUALIS B5: 6 pontos;
- Estrato QUALIS C ou sem classificação: 4 pontos.

Outros fatores adicionam pontos aos pedidos: coordenação de projeto de pesquisa com financiamento pelo requisitante (4 pontos), mérito reconhecido, com ou sem apoio financeiro, por uma agência de fomento (10 pontos com financiamento e 6 sem financiamento), divulgação em evento considerado importante para a área de Sistemas de Informação, tais como SBSI, ICEIS e AMCIS (6 pontos) e o número de artigos apresentados no mesmo evento (2 pontos, se apresentar mais de um artigo).

Após a contagem dos pontos, é possível chegar a um conceito final, que será utilizado para definir a cota de financiamento sugerida. O pedidos que obtiveram, no mínimo, 2 pontos, recebem o conceito *E*. Com 4 pontos, o pedido recebe o conceito *D*; com 8 pontos, recebe o conceito *C*; com 12 pontos, recebe o conceito *B;* e a partir de 16 pontos, recebe o conceito *A*. Com base no conceito atribuído ao pedido, a Comissão de Mérito sugere um valor financeiro de financiamento, que respeita a divisão entre eventos no Brasil e no exterior.

3.2 Casos de Uso de Sistema de Avaliação de Mérito

Os principais casos de uso do sistema de Avaliação de Mérito são mostrados no diagrama da figura 5. Em seguida, apresentamos breves descrições destes casos de uso.

Figura 5 – Diagrama de caso de uso

- **Logar:** este caso de uso permite autenticar um usuário no sistema, sendo pré-requisito para que este usuário execute outros casos de uso. Durante a execução deste caso de uso, o usuário deve informar seu login e sua senha de acesso;
- **Deslogar:** este caso de uso permite que um usuário remova sua autenticação do sistema, deixando de participar de outros casos de uso até que execute o caso de uso "Logar" com sucesso;
- **Trocar senha:** este caso de uso permite que um usuário autenticado troque a sua senha de acesso ao sistema. O usuário precisará digitar duas vezes a nova senha, evitando que digite erroneamente sua senha;

- **Editar perfil:** este caso de uso permite que o usuário edite as informações que o sistema armazena sobre ele. Estas informações incluem seu nome, e-mail, nome do projeto de pesquisa financiado que é coordenado pelo usuário, mês e ano de encerramento deste projeto de pesquisa;
- **Submeter pedido:** este caso de uso permite que um usuário autenticado submeta pedidos. O usuário precisa preencher os dados sobre o pedido, que incluem a cota a qual ele pertence, o nome do artigo, o nome da conferência, a qualificação da conferência, o pedido de financiamento e o número de artigos;
- **Deletar pedido:** este caso de uso permite que o usuário delete um pedido. Para isto, o usuário precisa ser o dono do pedido e este precisa estar no estado de *pendente*;
- **Consultar pedido:** este caso de uso permite que um usuário autenticado consulte um pedido;
- **Reprovar pedido:** este caso de uso permite que um usuário autenticado que seja membro da Comissão de Mérito rejeite um pedido. Para isto, o pedido precisa estar no estado de *avaliado*;
- **Aprovar pedido:** este caso de uso permite que um usuário autenticado que seja membro da Comissão de Mérito aprove um pedido. Para isto, o pedido precisa estar no estado de *avaliado*;
- **Encerrar pedido:** este caso de uso permite que um usuário autenticado que seja presidente da Comissão de Mérito encerre um pedido. Para isto, o pedido precisa estar no estado de *pendente* ou ter sido rejeitado pelos membros da Comissão de Mérito;
- **Encaminhar pedido:** este caso de uso permite que um usuário autenticado e que seja presidente da Comissão de Mérito encaminhe um pedido. Para isto, o pedido precisa ter sido aprovado pelos membros da Comissão de Mérito;

- **Manter professor:** este caso de uso permite que um usuário autenticado que seja presidente da Comissão de Mérito crie, delete e altere informações sobre os usuários cadastrados no sistema. Para criar novos usuários para o sistema é necessário informar um login que ainda não tenha sido usado;
- **Alterar configuração da Comissão de Mérito:** este caso de uso permite que um usuário autenticado e que seja presidente da Comissão de Mérito altere as configurações da Comissão de Mérito. Para isto, é necessário informar o login dos dois membros da Comissão de Mérito e o login do presidente;
- **Emitir relatório:** este caso de uso permite que um usuário autenticado e que seja presidente da Comissão de Mérito emita um relatório sobre um pedido. Para isto, o pedido precisa estar no estado de *encaminhado*.

3.3 Classes do Sistema de Avaliação de Mérito

Os conceitos do domínio da aplicação são representados pelas quatro classes apresentadas na figura 6.

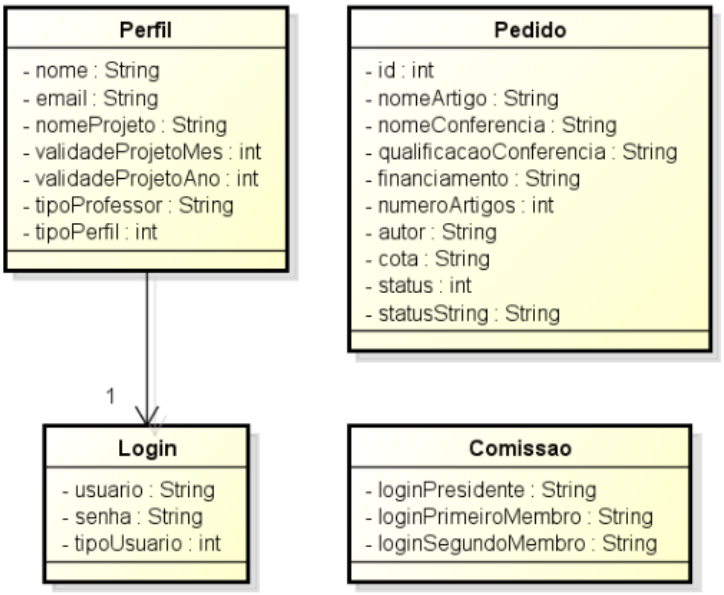

Figura 6 – Diagrama de classes do domínio

A classe “Perfil” guarda as informações sobre o usuário, como nome e email, além de uma referência para a classe “Login”, que contém as informações de autenticação do usuário. A classe “Pedido” possui as informações necessárias para criação dos pedidos submetidos e avaliados pela Comissão de Mérito. Por fim, a classe “Comissão” mantém a composição da Comissão de Mérito, indicando seu presidente e os dois professores que atuam como seus membros. A figura 7 apresenta o diagrama de classe do caso de uso “Trocar Senha”.

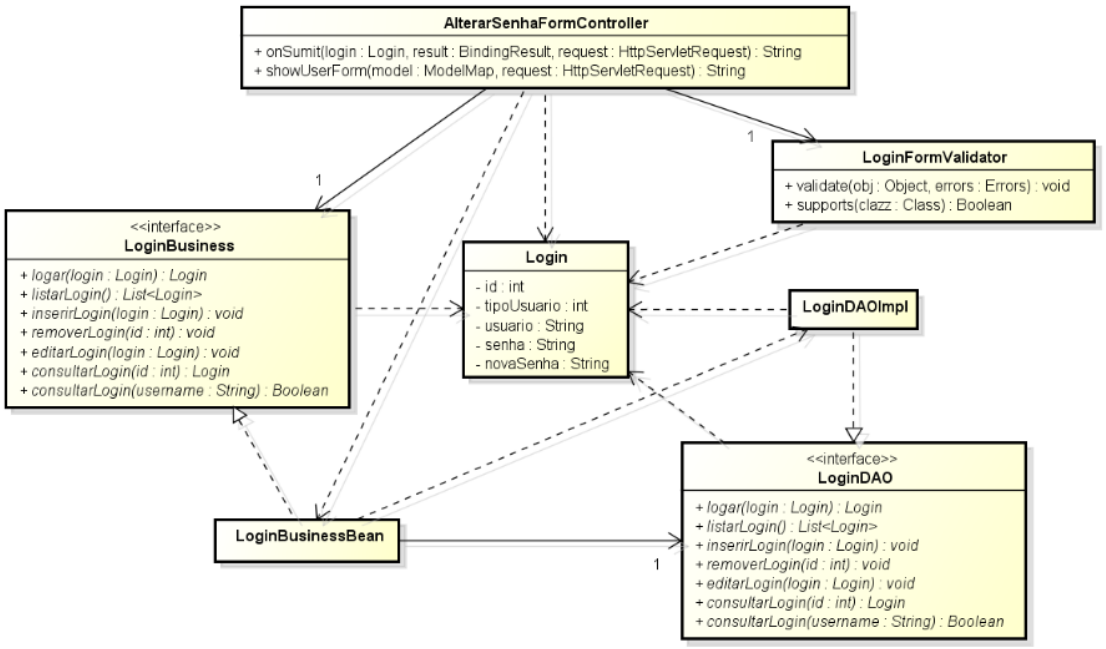

Figura 7 – Diagrama de classe do caso de uso Alterar Senha

A classe *Login* guarda as informações de autenticação do usuário, como sua identificação e sua senha. *AlterarSenhaFormController* é a classe de controle responsável pelo fluxo da transação. Ela instancia a classe *LoginBusinessBean* através de sua interface *LoginBusiness*. A classe *LoginBusinessBean* é responsável pelas regras de negócio de algumas transações do sistema, inclusive da troca de senha. Estas regras de negócio são derivadas dos requisitos do sistema e impõem restrições à maneira como este deve ser utilizado. A classe *LoginBusinessBean* também define a classe que deve acessar o banco o dados que, neste caso, é a classe *LoginDAOImpl*, que é instanciada

através de sua interface, a *LoginDAO*. A *LoginDAOImpl* possuiu as informações necessárias para acessar o banco de dados e fazer as alterações requisitadas.

A classe *LoginFormValidator* é a responsável por validar as informações inseridas pelo usuário. Ela é chamada pela classe de controle antes que qualquer ação seja feita. A figura 8 mostra o diagrama de seqüência do caso de uso "Trocar Senha".

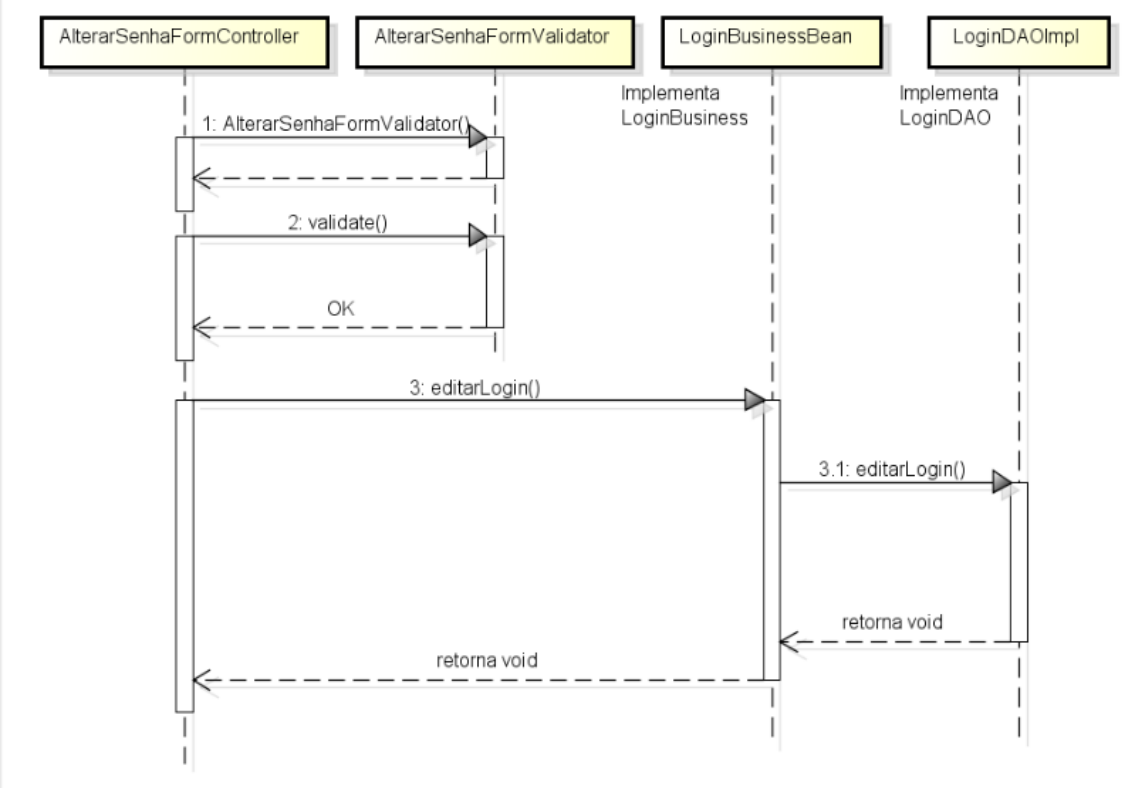

Figura 8 - Diagrama de seqüência do caso de uso Alterar Senha

3.4 O Banco de Dados do Sistema de Avaliação de Mérito

O modelo do banco de dados do Sistema de Avaliação de Mérito é apresentado na figura 9. O modelo é definido em torno de duas entidades: *Usuário* e *Pedido*. A entidade *Usuário* possui as informações referentes ao perfil do usuário no sistema. Ela possui três chaves estrangeiras: uma para a entidade *Login*, que guarda as informações de login do usuário; uma para *Tipo_Usuario*, que identifica o papel do usuário no sistema, que pode ser presidente, membro da comissão ou usuário comum; e uma para

Tipo_Professor, que guarda as informações sobre o perfil do usuário perante o DIA/UNIRIO, sendo as opções “ativo e colaborador”, “ativo” e “inativo”.

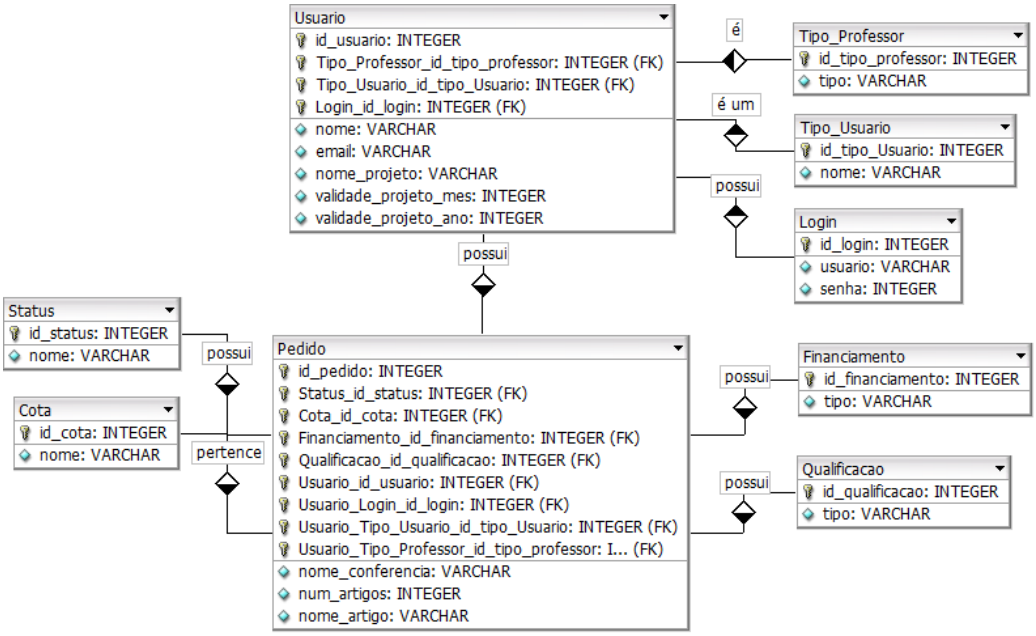

Figura 9 – Diagrama de classe do banco de dados

A entidade *Pedido* engloba todas as informações sobre os pedidos feitos no sistema. Ela possui quatro chaves estrangeiras. A primeira é para a entidade *Status*, que representa o estado do pedido. O estado de um pedido pode ser: pendente, quando o pedido ainda não foi aprovado pelo presidente; avaliado, quando o presidente envia o pedido para avaliação dos membros da Comissão de Mérito; encaminhado, quando o pedido é aprovado pelos membros e homologado pelo presidente; rejeitado, quando o pedido é rejeitado pelo presidente na avaliação inicial; e encerrado, quando o pedido é reprovado pelos membros da Comissão e homologado pelo presidente. Ainda há os estados intermediários entre a aprovação do presidente e o resultado final do processo, conforme apresentado na figura 4.

A segunda chave estrangeira da entidade *Pedido* é para a entidade *Qualificação*. Essa entidade possui os tipos de qualificação que um pedido pode ter, de acordo com a

classificação Qualis do evento em que o artigo científico foi publicado, a saber: A1, A2, B1, B2, B3 ,B4, B5, C e não definida.

A terceira chave é para entidade *Financiamento,* que contém os tipos de financiamento externo que um pedido submetido à Comissão de Mérito pode ter: “mérito com financiamento”, “mérito sem financiamento” e “nenhum”.

Finalmente, a quarta chave da entidade *Pedido* é para entidade *Cota*, que guarda os tipos de cota que podem ser utilizadas quando um pedido é submetido à Comissão de Mérito. Como foi explicado anteriormente, os professores têm uma cota de três pedidos anuais, sendo dois para eventos nacionais e um para evento internacional. Essa entidade define a que cota o pedido associado pertence.

3.5 Considerações Finais

Neste capítulo, foi explicado o Processo de Avaliação de Mérito para concessão de financiamento para artigos científicos do DIA/UNIRIO em cada uma de suas etapas, mostrando o papel de cada membro da Comissão de Mérito. Também foram apresentados os modelos de caso de uso, de classes e do banco de dados utilizados na aplicação.

O próximo capítulo falará sobre a implementação do Sistema de Avaliação de Mérito. Serão apresentadas as funcionalidades da aplicação do ponto de vista de cada usuário envolvido com ela. Também será feita uma discussão sobre o uso do Spring MVC no desenvolvimento da aplicação, identificando suas vantagens e desvantagens.

4 A Implementação do Sistema de Avaliação de Mérito do DIA/UNIRIO

Este capítulo apresenta as principais funcionalidades do sistema de apoio às atividades da Comissão de Mérito do DIA/UNIRIO, como o processo de submissão de pedidos e seu trânsito entre os membros da Comissão. São focadas as principais ações que os usuários do sistema podem executar. O capítulo também identifica as vantagens e desvantagens percebidas pelo uso do Spring MVC na implementação da aplicação, apontando suas facilidades e eventuais dificuldades no seu desenvolvimento.

4.1 Funcionalidades do Professor

Um professor é um usuário comum do sistema. Ele pode alterar suas informações, como senha e perfil, além de submeter pedidos. A figura 10, apresentada anteriormente, mostra a tela de login que todos os membros devem acessar para se autenticar. A figura 11 apresenta a tela inicial de um professor.

Figura 10 – Tela de Login

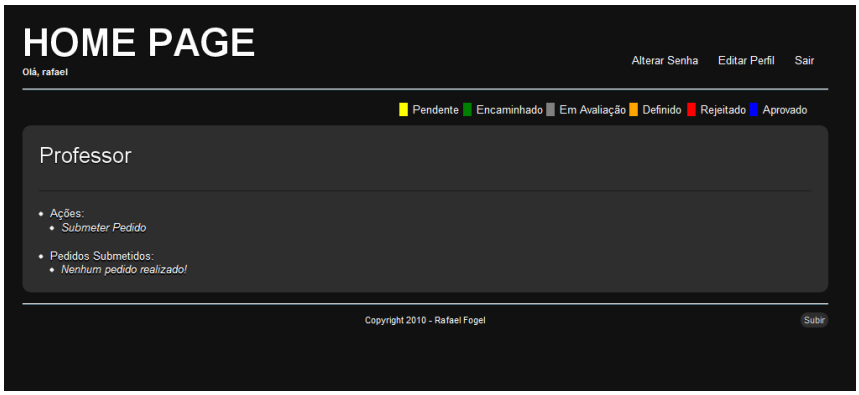

Figura 11 – Tela inicial de um professor

Clicando em *Submeter Pedido*, o usuário será direcionado para a tela de submissão de pedidos, mostrada na figura 12.

Figura 12 – Tela de submissão de pedido

Quando um usuário submete um pedido, o mesmo é mostrado na tela inicial, colorido de acordo com o seu estado. Caso o usuário deseje deletar um pedido, basta selecioná-lo na tela inicial e clicar em *Deletar*. Porém, o pedido precisa estar no estado *Pendente* para que possa ser removido pelo usuário que o submeteu. A figura 13 mostra a tela de informações do pedido.

Figura 13 – Tela de informações do pedido

As figuras 14 e 15 mostram as telas de edição do perfil do usuário e alteração de senha, respectivamente.

Figura 14 – Tela de edição de perfil

Figura 15 – Tela de alteração de senha

A figura 14 mostra um exemplo de edição do perfil do usuário. É necessário

preencher todas as informações. Caso o usuário não participe de nenhum projeto de pesquisa, os campos de validade do projeto devem ficar em branco. Na figura 15, é mostrado um exemplo de alteração de senha. O campo de confirmação de senha serve para o usuário não digite erronamente a sua senha.

4.2 Funcionalidades do Avaliador de Mérito

Os membros da Comissão de Avaliação de Mérito podem executar as mesmas ações disponíveis para os professores, com a diferença de possuírem uma área exclusiva com comandos para avaliação dos pedidos aceitos pelo presidente da Comissão. A tela inicial de um membro da Comissão de Mérito é apresentada na figura 16.

Figura 16 – Tela inicial do 1º membro da Comissão de Mérito

A tela de informações do pedido é a mesma apresentada aos professores, porém com dois botões extras, um para aceitar e outro para rejeitar o pedido. Toda vez que o presidente da Comissão aceita um pedido, este é automaticamente mostrado para os membros da Comissão de Mérito. Enquanto ele estiver sendo avaliado, será apresentado na tela inicial. O pedido só não será mais apresentado quando a sua situação for definida, ou seja, quando for rejeitado ou aceito pelo presidente.

4.3 Funcionalidades do Presidente da Comissão de Mérito

O presidente da Comissão de Mérito, assim como os membros, possui as mesmas funcionalidades de um professor e uma área exclusiva para gerenciar os pedidos. Nessa área, são apresentados todos os pedidos do site, independente do seu estado no processo de avaliação. A figura 17 mostra a tela inicial do presidente da Comissão de Mérito.

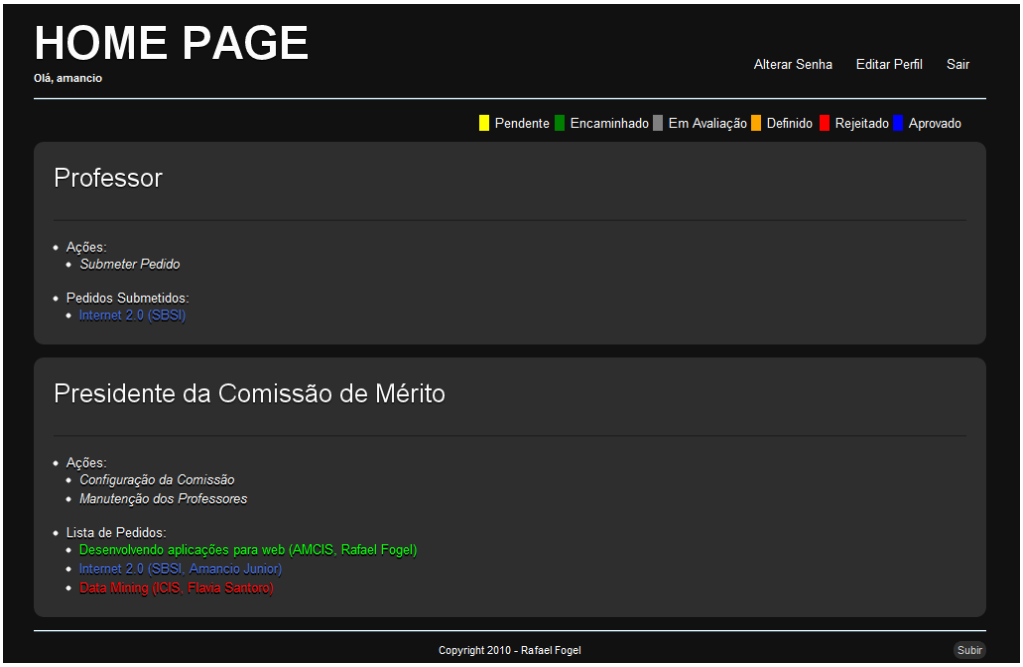

Figura 17 – Tela inicial do presidente da Comissão de Mérito

O presidente ainda é responsável por administrar a composição da Comissão de Mérito. É ele quem indica, no contexto do sistema, quem são os membros avaliadores e quem será o próprio presidente, caso este esteja de saída do cargo. A figura 18 mostra a tela de configuração da Comissão de Mérito.

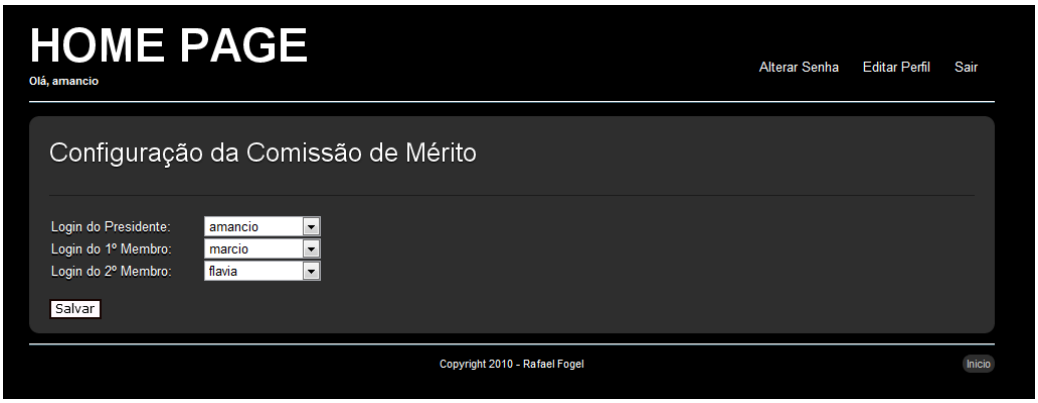

Figura 18 – Tela de configuração da Comissão de Mérito

Assim como a composição da Comissão de Mérito, o presidente também é responsável pela manutenção dos usuários do site. Só o presidente pode adicionar e remover usuários. A figura 19 mostra a tela onde os usuários são criados e removidos. Esta tela mostra a lista de usuários cadastrados, com seu login, nome e e-mail.

Figura 19 – Tela de manutenção de usuários

Figura 20 – Tela de atualização do usuário

A figura 20 mostra a tela de atualização do usuário. É nela que o presidente preenche os dados de um novo usuário ou edita os dados de um usuário já existente.

4.4 Discussão sobre o uso do Spring

O uso do Spring MVC no desenvolvimento do projeto foi considerado muito satisfatório. Apesar de não existir muita documentação na Internet, já que o Spring MVC ainda é um framework recente e em crescimento, ele mostrou-se completo para qualquer tipo de aplicação e motivo de formação de uma comunidade disposta a ajudar os novos desenvolvedores.

O uso das anotações disponibilizadas pelo framework reduz bastante o esforço de codificação e oferece ao programador liberdade na hora de desenvolver as classes, já que permite métodos com nomes e parâmetros personalizados. Seu uso também facilita a leitura do código, pois as anotações possuem uma sintaxe de fácil compreensão.

O Spring MVC ainda possui outras vantagens. Uma delas, muita usada entre os desenvolvedores, é a possibilidade de se integrar com outros frameworks, tais como Struts e Java Server Faces, dando liberdade ao desenvolvedor para usar as vantagens de cada framework.

Outro fator que estimula o uso do Spring MVC é o uso da técnica de inversão de controle, que permite o desenvolvedor passar para o framework a responsabilidade de instanciar objetos passados como parâmetro e atribuí-los às variáveis da classe. Essa técnica é muita usada principalmente em construtores, reduzindo o acoplamento entre as classes.

Por outro lado, as anotações tendem a confundir o programador a medida que o código vai crescendo. Como as informações das classes das camadas de controle e visão estão inseridas nelas próprias, o programador pode se confundir em suas requisições entre classes. Usando a forma padrão do Spring MVC, ou seja, sem anotações, as informações sobre as classes ficam guardadas em um único arquivo, o *dispatcher-servlet.xml*, facilitando o controle pelo desenvolvedor.

Para finalizar, a grande dificuldade do uso do Spring MVC foi a falta de informação. Como foi dito anteriormente, o Spring MVC ainda está em crescimento e existe pouca documentação na Internet, além da oficial. Porém, a comunidade dos fóruns oficiais é bem receptiva com novos usuários e, com um pouco de paciência, o usuário consegue sair com a sua dúvida sanada.

4.5 Considerações Finais

Neste capítulo, foram apresentadas as funcionalidades de cada usuário do sistema, juntamente com as telas que servem de exemplo sobre o seu uso. Também foi feita uma breve explicação de como se utilizar o site e como ocorrem os processos de submissão e avaliação dos pedidos pelos integrantes da Comissão de Mérito. O próximo capítulo apresentará a conclusão deste projeto, suas limitações e as perspectivas para a realização de trabalhos futuros.

5 Conclusão

5.1 Contribuições

Neste trabalho foi criado um sistema para gestão e controle de pedidos de avaliação mérito de artigos acadêmicos do DIA/UNIRIO. Esse sistema é composto por um portal online que administra os artigos e os organiza de forma simples para os seus responsáveis. Através desse sistema, os membros e o presidente da Comissão de Mérito podem avaliar os pedidos de publicação de artigos de forma rápida e eficaz.

Este trabalho também serviu para demonstrar o uso do Spring MVC, um framework recente e em crescimento e que, apesar de ainda desconhecido por muitos, vem ganhando espaço no mercado de trabalho.

5.2 Trabalhos Futuros

Visando uma comunição mais rápida entre os integrantes da Comissão de Mérito, a primeira etapa é da criação de um sistema de envio de mensagens. Toda vez que algum membro avalia um pedido ou quando o presidente aprova um pedido, o sistema pode comunicar os demais integrantes, avisando-os que sua participação no sistema é requisitada.

A tabela de conferências e suas qualificações pode ser incluída na página de submissão de pedidos, facilitando a vida do usuário no preenchimento dos dados pedido.

Os critérios para publicações e regras para cálculo do valor do financiamento dos pedidos estão inseridos no código. No futuro, se estes itens mudarem, o código terá que ser alterado. Sendo assim, um possível trabalho futuro é a criação de um meio para que o sistema consiga registrar novas regras e critérios, dispensando a alteração do código.

Outra possível atualização é a migração do sistema para um ambiente de desenvolvimento baseado em nuvem, como o Google App Engine [GOOGLE, 2011]. O sistema hospedado na nuvem dispensa o uso de servidores, que requerem um aparato mínimo de funcionamento, além de manutenção.

A implementação do banco de dados pode ser melhorada, já que, por motivos de falta de tempo, o projeto foi feito usando o PostgresSQL 8.1, que não permite o uso de stored procedures, apenas de funções. Com isso, algumas chamadas ao banco de dados estão inseridas no código do sistema, não sendo o ideal para este tipo de projeto.

Referências

ZEMEL, Társio, "*O que é um framework? Definição e benefícios de se usar um framework*". Disponível em http://codeigniterbrasil.com/passos-iniciais/o-que-e-um-framework-definicao-e-beneficios-de-se-usar-frameworks/, acessado em 10/05/2011(a).

ZEMEL, Társio, "*MVC (Model – View - Controller)*". Disponível em http://codeigniterbrasil.com/ passos-iniciais/mvc-model-view-controller/, acessado em 08/05/2011(b).

MACORATTI, J.C., "*Padrões de Projeto : O modelo MVC - Model View Controller*". Disponível em http://www.macoratti.net/vbn_mvc.htm, acessado em 10/05/2011.

SPRINGSOURCE. Disponível em http://www.springsource.org/, acessado em 10/02/2011.

JAVABEAT, "*Controllers in Spring MVC*". Disponível em http://www.javabeat.net/tutorials/259-controllers-in-spring-mvc.html, acessado em 08/05/2011.

GRONER, L., "*XML: como posso usar: vantagens e desvantagens – Introdução ao XML: Parte II.* Disponível em http://www.loiane.com/2009/02/xml-como-posso-usar-vantagens-e-desvantagens-introducao-ao-xml-parte-ii/, acessado em 10/05/2011.

INFOBLOG, "*XML Vs Anotação Vs Convenção*". Disponível em http://infoblogs.com.br/view.action?contentId=27458&XML-VS-Annotations-VS-Convencao.html, acessado em 19/02/2011.

STRUTS, The Apache Foundation. Disponível em http://struts.apache.org/ acessado em 02/01/2011.

EDUARDO, C., "*JAVA – Struts*". Disponível em http://imasters.com.br/artigo/3372/java/struts/, acessado em 02/03/2011.

FOWLER, M., "*Inversion of Control*". Disponível em http://martinfowler.com/bliki/InversionOfControl.html, acessado em 18/02/2011.

FACES, The Apache Foundation. Disponível em http://myfaces.apache.org/, acessado em 08/05/2011.

HALL, MARTY, "*JSF Introduction, 2005*". Disponível em http://www.apl.jhu.edu/~hall/, acessado em 11/05/2011.

JOHNSON, ROD, "*Expert One-on-One: JEE Design e Development*", Peer Information; 1st edition, October 2002.

JAVAFACES, "*JavaServer Faces Technology*". Disponível em http://www.oracle.com/technetwork/java /javaee/javaserverfaces-139869.html, acessado em 10/05/2011.

GOOGLE, "*Google App Engine*". Disponível em http://code.google.com/intl/pt-BR/appengine/, acessado em 11/05/2011.