MAC 0332
Engenharia de Software
SI para grupos de pesquisa
Architecture Notebook

6 de novembro de 2011

1 Propósito

Este documento descreve a filosofia, as decisões, as restrições, as justificativas, os elementos significativos e qualquer outro aspecto importante do sistema que dão forma ao seu *design* e a sua implementação.

2 Objetivos arquiteturais e filosofia

A arquitetura desse projeto visa ao desenvolvimento de um sistema organizado, durável e prático, tanto para desenvolvedores, quanto para usuários.

Por meio de ferramentas que abstraem implementações de recursos normalmente complexos (como banco de dados e interfaces *Web*), podemos, de certa forma, limitar o escopo de desenvolvimento à parte da lógica do sistema.

Mediante esta decisão arquitetural e de projeto, simplificamos o desenvolvimento e reduzimos em grande parte o volume de código necessário, o que facilitará sobremaneira sua documentação, verificação e manutenção.

Logo, os principais objetivos arquiteturais serão garantir que:

1. Estas ferramentas sejam devidamente aplicadas no sistema, isto é, que elas estejam corretamente instaladas e instanciadas, e que nosso sistema as use e as faça comunicarem-se entre si de maneira bem sucedida, aproveitando ao máximo suas facilidades;
2. A lógica do sistema esteja de acordo com os requerimentos do cliente, priorizando sua testabilidade e manutenabilidade, uma vez que a maioria das tarefas tecnicamente complexas estará sendo coberta pelas ferramentas auxiliares.

3 Suposições e dependências

3.1 Dependências tecnológicas do projeto:

- Um servidor Tomcat, com a versão 6, para hospedar nosso sistema;
- Um banco de dados MySQL, para persistir os dados gerenciados pelo sistema;
- a JDK e o Eclipse (Enterprise Edition ou equivalente), para desenvolvimento do sistema usando Java para Web;
- Um navegador para visualizarmos a interface Web do sistema;
- A biblioteca JDBC e seu plugin para banco de dados MySQL, para viabilizar a conexão da aplicação Java com o banco de dados;

- A implementação Hibernate da JPA, para abstrair o banco de dados por meio de um mapeamento objeto-relacional, facilitando o desenvolvimento do sistema em Java, que é uma linguagem orientada a objetos;
- O *framework* VRaptor, que implementa o controle da arquitetura MVC, para controlar e comunicar facilmente a aplicação Java com a interface Web do sistema;
- As bibliotecas JUnit e Mockito para execução de testes automatizados do sistema;
- As bibliotecas JQuery e JQueryUI para interatividade nas páginas Web;
- O sistema Git para controle de versão e concorrência do código do projeto;
- Quaisquer outras dependências indiretas geradas pelas tecnologias citadas.

3.2 Dependências de conhecimento e experiência da equipe:

- Programação em linguagem Java usando a IDE Eclipse;
- Análise, Modelagem e Desenvolvimento com orientação a objetos;
- Produção de páginas Web em XHTML (com CSS);
- Competência para desenvolvimento de páginas dinâmicas Web com a linguagem Java, mediante o uso de *Expression Language* e JSTL (taglibs).
- Modelagem e projeto de bancos de dados;
- Competência para desenvolvimento com as bibliotecas Hibernate, VRaptor, JQuery, JUnit e Mockito;
- Competência para desenvolvimento com o sistema de controle de versão e concorrência Git.

3.3 Suposições:

- Assumimos que as ferramentas, bibliotecas e demais tecnologias usadas no desenvolvimento do projeto são funcionais, isto é, agem como o esperado, segundo os registros de suas respectivas documentações, dispensando, por assim dizer, testes adicionais.

4 Requisitos para realizar a arquitetura

Para que se possa implementar a arquitetura descrita neste documento, a equipe de desenvolvimento deve ter os seguintes recursos devidamente instalados e funcionando:

- Ambiente de desenvolvimento Java usando *Eclipse Enterprise Edition* (ou equivalente);
- Um servidor Tomcat devidamente configurado para rodar a partir do Eclipse;

- Um repositório remoto Git, onde serão mantidos o código fonte do projeto e os diversos artefatos importantes que forem produzidos ao longo do projeto (inclusive este);
- Um projeto em branco de VRaptor do site da Caelum importado no Eclipse. Este projeto servirá como gabarito base para o aplicativo Java que conterá a lógica do sistema. Este projeto deve ser adicionado ao repositório Git, preferencialmente por meio do *plug-in* Git do Eclipse. Feito isto, todos os membros da equipe poderão sincronizar seus repositórios locais ao repositório remoto, com o objetivo de unirem-se ao desenvolvimento do sistema (ou seja, só uma pessoa precisa realizar esta tarefa e, em seguida, todos poderão importar este projeto a partir do repositório remoto Git, usando o Eclipse);
- Os JAR's das bibliotecas JDBC, Hibernate (com *annotations*), JUnit (versão 4) e Mockito, bem como suas respectivas dependências, incluídas na pasta WebContent/WEB-INF/lib do projeto Java. A biblioteca VRaptor já está inclusa no projeto em branco da Caelum. (Do mesmo modo que no item anterior, uma vez que alguém tenha conseguido fazer isto, basta usar o repositório remoto Git para que todos obtenham suas cópias locais do projeto, igualmente configuradas.)

5 Decisões, restrições e justificativas

- O sistema será programado em Java, pois é uma linguagem orientada a objetos, guarnecida de diversas facilidades e recursos que auxiliam tanto no desenvolvimento, quanto na manutenção do sistema. Além disso, a profícua comunidade desta linguagem provê diversas ferramentas, as quais facilitam a aplicação de Java na construção de sistemas de software pertinentes às exigências do mercado e dos círculos de pesquisa.
- O ambiente de desenvolvimento será o *Eclipse Enterprise Edition* (ou equivalente), pois é equipado com todas as ferramentas básicas para desenvolvimento para Web, usando Java. Em particular, ele oferece a possibilidade de rodar um servidor Tomcat local no próprio Eclipse, o que possibilita que os desenvolvedores possam testar o sistema em suas próprias máquinas a qualquer momento, como se o estivessem rodando no servidor remoto.
- Para comunicar nossa aplicação Java com a interface Web do sistema, usaremos o *framework* VRaptor. Como esta implementa o controlador da arquitetura MVC, ela simplifica drasticamente a construção de controladores, além de gerenciar automaticamente as instâncias das diversas partes do sistema, criando-as e mantendo-as disponíveis conforme demandas solicitadas pelo cliente Web.
- Para comunicar nossa aplicação Java com o banco de dados, usaremos a biblioteca Hibernate, que, por sua vez, se conectará ao banco de dados por meio da biblioteca JDBC. Devido ao mapeamento objeto-relacional proporcionado pelo Hibernate (de acordo com a especificação JPA), prescindimos da complexidade da criação manual das *queries* ao banco de dados, haja vista que esta tarefa será realizada automaticamente. O emprego deste *framework* permite que a programação orientada a objetos para banco de dados seja realizada de maneira quase que natural, propiciada pelo Hibernate que viabiliza a correspondência entre classes e relações.

- Para realizar os testes de unidade, usaremos as bibliotecas JUnit e Mockito. A JUnit é uma das ferramenta para testes de unidade em Java largamente utilizada. Além disso, o Eclipse tem *plug-ins* que auxiliam seu uso. A Mockito possibilita a simulação de objetos e respectivos comportamentos, possibilitando a realização de testes de unidade, que de fato não dependam de outras unidades para serem testados.

- Para facilitar o desenvolvimento das páginas Web, usamos a biblioteca JQuery, caracterizada pela facilidade de uso e por disponibilizar ferramentas poderosas, com suporte a diversos navegadores. Isto permite que os desenvolvedores concentrem-se no desenvolvimento de novas funcionalidades, em vez de diferenças entre navegadores.

- Para elementos de interface Web, a JQuery UI fornece diversos recursos avançados, por meio de comandos simples. É possível criar rapidamente calendários, telas com abas ou até mesmo formulários dinâmicos.

6 Mecanismos arquiteturais

Esta é uma das partes que evoluirão ao longo do projeto, moldando-se de acordo com as necessidades que forem surgindo.

6.1 Controlador

Módulo responsável por controlar a parte lógica, sob a interface Web.

Propósito: Fornecer as páginas web pedidas pelo usuário, assim como executar qualquer lógica de alguma página em particular, como por exemplo, o cadastro de um usuário.

Funcionamento: Cada endereço *URI* solicitado pelo usuário deve estar associado a algum método de algum controlador, assim chamado, passando os dados que o usuário enviou juntamente com a requisição. Este método irá processar o pedido, redirecionando o resultado para uma página web.

6.2 Modelo - *DAO - Data Access Object*

Módulo responsável por armazenar e fornecer acesso a um certo tipo de dado do banco de dados.

Propósito: Encapsular o Hibernate, integrando-se melhor ao sistema e facilitando testes e manutenção.

Funcionamento: Por ser uma classe do nosso próprio sistema, é mais facilmente simulada (*mocked*) e integrada ao VRaptor. Cada método é basicamente uma chamada direta ao Hibernate, sendo que algumas também realizam *Transactions*.

6.3 Injeção de Dependência

Método utilizado para lidar com dependências de uma classe com outra. Sempre que uma classe precisar de um objeto de outra, deverá dar-se preferência ao uso de injeções de dependência.

Propósito: reduzir o acoplamento a classes de outras bibliotecas, melhorar a legibilidade e a testabilidade do código.

Funcionamento: Quando o VRaptor instanciar um controle, aquele verificará que o controle depende de um objeto de acesso a dados (DAO) e criará uma instância deste. Isto é possível, pois o objeto de acesso a dados fora devidamente anotado, para que o VRaptor tenha conhecimento de que, doravante, deverá gerenciar as instâncias deste objeto.

6.4 *Factories*

Módulo que facilita a criação de objetos com configurações pré-determinadas. Compatível com injeção de dependências.

Propósito: Fornecer objetos prontos para o uso.

Funcionamento: Uma classe com diversos métodos, onde cada um devolve um dos objetos desejados já configurados.

6.5 Testes de unidade

Método utilizado para testar isolada e individualmente unidades do sistema.

Propósito: Garantir que cada classe do projeto funcione individualmente.

Funcionamento: Para cada classe do projeto (excluindo aquelas que apenas encapsulam uma biblioteca externa), cria-se uma classe que implementa um teste de unidade, utilizando o JUnit. Neste tipo de teste, as saídas dos métodos são comparadas a valores esperados.

7 Abstrações chave

Interface Web: conjunto de páginas *web* que compõem o *Website* do sistema e que podem ser visualizadas pelos usuários. Envia requisições para o sistema, conforme ações dos usuários e exibe, ao usuário, as informações que o sistema lhe forneceu em resposta.

Banco de dados: conjunto de dados armazenados em um servidor, conforme um modelo relacional (usando MySQL). Os dados armazenados dizem respeito aos grupos de pesquisa, usuários, publicações e demais informações que foram passadas ao sistema de grupos de pesquisa.

Lógica do sistema: aplicativo (Java) que gerencia a lógica do sistema, interagindo com os usuários através da Interface Web e persistindo os dados obtidos no Banco de Dados.

8 Camadas da arquitetura

Há, no total, seis camadas na arquitetura do projeto. A primeira, e mais externa, é a interface Web, composta pelos diversos arquivos `*.jsp`, `*.css` e `*.js` que têm sido escritos, formando as diversas páginas Web exibidas aos usuários quando do acesso ao sistema. A segunda camada é o controle da arquitetura MVC, iimplementada pelo VRaptor, que serve para ligar a primeira camada com a terceira, ou seja, com as classes de controle do sistema. Estas, por sua vez, comunicam-se com a quarta camada, a saber, a lógica do sistema. É nesta, que o trabalho da equipe tem se concentrado durante o desenvolvimento do sistema. A quinta camada é composta pelos *Data Access Objects* (DAOs), que viabiliza o armazenamento de dados e permite que a quarta camada (lógica do sistema) se comunique com a sexta e última camada, que é o banco de dados. Este, por sua vez, é abstraído pelo mapeamento objeto-relacional do Hibernate.

9 Visões arquiteturais

- **Visão lógica:**

- **Casos de uso com requisitos arquiteturalmente importantes:**
  Um relato completo dos casos de uso levantados pela equipe estão no documento "Modelo de Casos de Uso", que acompanha esta documentação.

  Alguns casos considerados *particular ou exemplarmente* relevantes com relação à arquitetura do sistema – por exemplo, por nos forçarem a garantir certas relações diretas ou indiretas entre módulos ou unidades do sistema, ou a realização de certas estruturas de interface com o usuário, são:

  - Administrador cadastra um grupo de pesquisa;
  - Administrador cadastra um contribuinte e o filia a um grupo;
  - Administrador aponta um contribuinte como autor de uma publicação;
  - Contribuinte cadastra um projeto em uma linha de pesquisa que estuda;
  - Contribuinte cadastra uma publicação e cita quais os projetos aos quais ela se aplica;
  - Usuário realiza login;
  - Visitante lista disciplinas oferecidas por um grupo;
  - Visitante lista grupos e linhas de pesquisa;
  - Visitante lista projetos oferecidos por uma linha pesquisa;
  - Visitante lista projetos oferecidos por um grupo.
  - Visitante lista Contribuintes de um projeto, Linha ou um Grupo
  - Visitante lista Publicações de um projeto, Linha ou um Grupo