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

24 de outubro 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 à 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.

Através 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.

Teremos menos preocupações e o volume de código será bastante reduzido, facilitando sua documentação, verificação e manutentção.

Logo, os principais objetivos arquiteturais serão garantir que:

1. Essas ferramentas sejam devidamente aplicadas no sistema. Isso é, que elas estejam corretamente instaladas e instanciadas, e que nosso sistema as use e as faça se comunicarem entre si de maneira bem sucedida e aproveitando ao máximo suas facilidades;
2. A lógica do sistema esteja de acordo com os requerimentos do cliente, usando uma estrutura o mais simples possível de testar, uma vez que a maioria das complicações técnicas 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 desenvolvermos nossos sitema usando Java para Web;
- Um navegador para visualizarmos a interface Web do nosso sistema;
- A biblioteca JDBC, junto de seu plugin para MySQL, para comunicar nossa aplicação Java com o banco de dados;
- A implementação Hibernate da JPA para abstrair o banco de dados através de um mapeamento objeto-relacional, facilitando o desenvolvimento do sistema em Java, que é uma linguagem orientada a objetos;

- A bilbioteca VRaptor, para comunicarmos facilmente nossa aplicação Java com a interface Web do sistema;
- As bibliotecas JUnit e Mockito para testarmos nosso sistema;
- As bibliotecas JQuery e JQueryUI para interatividade nas páginas Web;
- Git para controle de versão do projeto;
- Quaisquer outras dependências indiretas geradas pelas tecnologias citadas.

3.2 Dependências de conhecimento e experiência da equipe:

- Programar em Java usando o Eclipse;
- Desenvolver com orientação a objetos;
- Produzir páginas Web em HTML (com CSS);
- Modelar bancos de dados;
- Saber usar as bibliotecas Hibernate, VRaptor, JQuery, JUnit e Mockito;
- Saber usar o controle de versão Git.

3.3 Suposições:

- Assumimos que as ferramentas, bibliotecas e demais tecnologias usadas no desenvolvimento do projeto são funcionais, isto é, agem do modo como suas respectivas documentações ditam e, portanto, não precisam ser testadas.

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 configurado para rodar a partir do Eclipse;
- Um repositório Git onde serão mantidos o código fonte do projeto e os diversos documentos importantes que forem produzidos ao longo do projeto (inclusive este);
- Um projeto em branco de VRaptor do site da Caelum importado no Eclipse. O aplicativo Java que conterá a lógica do nosso sistema será feito em cima dessa base. Esse projeto deve ser adicionado ao repositório Git, de preferência usando o *plug-in* Git do Eclipse. Uma vez feito isso, todos da equipe poderão se sincronizar com o repositório para se unir ao desenvolvimento do sistema (ou seja, só uma pessoa precisa fazer isso e depois todos poderão importar esse projeto do 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 vem incluída no projeto em branco da Caelum. (Do mesmo modo que no item anterior, uma vez que alguém tenha conseguido fazer isso, é só usar o repositório Git para que todos já tenham 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 que possui diversas facilidades e recursos que auxiliam tanto no desenvolvimento quanto na manutenção do sistema. Além disso, há na comunidade diversas ferramentas disponíveis que 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 ele vem com todas as ferramentas básicas para se programas 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 verdadeiro.

- Para comunicar nossa aplicação Java com a interface Web do sistema, usaremos a bilioteca VRaptor. 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 exigirem as demandas enviadas pelo cliente Web.

- Para comunicar nossa aplicação Java com o banco de dados, usaremos a biblioteca Hibernate, que se comunicará, por sua vez, com a biblioteca JDBC. Devido ao mapeamento objeto-relacional proporcionado pelo Hibernate (de acordo com a JPA), não haverá necessidade de nos preocuparmos com a criação manual das *queries* ao banco de dados, pois isso será feito automaticamente. Pode-se programar quase que naturalmente usando orientação a objetos, onde classes corresponderão, aproximadamente, a relações.

- Para realizar testes de unidade nas nossas classes, usaremos as bibliotecas JUnit e Mockito. a JUnit é a ferramenta padrão de testes em Java, e o Eclipse tem *plug-ins* que auxiliam seu uso. A Mockito servirá para que possamos simular objetos, impondo-lhes um comportamento pré-definido e, assim, podermos realizar 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, que possui uma facilidade de uso assim como ferramentas poderosas assim como suporte a diversos navegadores, permitindo os desenvolvedores focarem seu tempo em novas funcionalidades e não em diferenças entre navegadores.

- Para elementos de interface Web, a JQuery UI fornece diversos recursos avançados através de comandos simples. É possível criar um calendário, telas com abas ou até mesmo formulários dinâmicos com pouco esforço.

6 Mecanismos arquiteturais

Essa 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 por trás da 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 pedido pelo usuário deve estar associado com algum método de algum controlador, que será então chamado, passando os dados que o usuário enviou junto da requisição. Esse método irá processar o pedido, e terminar enviando uma página web.

6.2 *Data Access Object*

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

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

Funcionamento: Por ser uma classe do nosso próprio sistema, é mais facilmente mockada e integrado ao VRaptor. Cada método é basicamente uma chamada direta ao Hibernate, com algumas também utilizando 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:

Funcionamento:

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 unidades individuais do sistema de maneira isolada.

Propósito: Garantir de cada classe do projeto funciona individualmente.

Funcionamento: Para cada classe do projeto (excluindo aquelas que apenas encapsulam uma biblioteca externa), cria-se um teste utilizando o JUnit. Nesse teste, verifica-se se cada método dessa classa devolve os 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 de acordo com as ações dos usuários e mostra ao usuário as informações que o sistema lhe forneceu em resposta.

Banco de dados conjunto de dados armazenados em um servidor seguindo um modelo relacional (usando MySQL, no caso). 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 faremos, formando as diversas páginas Web que os usuários verão ao acessar nosso sistema. A segunda camada é o VRaptor, que serve para ligar a primeira camada com a terceira, que são os controladores. Estes, por sua vez, se comunicam com a quarta camada que é a que compõe a lógica do sistema. É nela que será focada nossa atenção e trabalho para construir o sistema.

A quinta camada será composta pelos *Data Access Objects* (DAOs), e servirá para a quarta camada (lógica do sistema) se comunicar com a sexta e última que é o banco de dados em si, só que 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.