UNIVERSIDADE FEDERAL DE JUIZ DE FORA - UFJF
PÓS GRADUAÇÃO EM DESENVOLVIMENTO DE SISTEMAS COM TECNOLOGIA JAVA

AUGUSTO CÉSAR BEVILAQUA

UMA COMPARAÇÃO DE DESENVOLVIMENTO WEB ENTRE A TECNOLOGIA SERVLETS / JSP E O FRAMEWORK VRAPTOR

JUIZ DE FORA
2013

AUGUSTO CÉSAR BEVILAQUA

UMA COMPARAÇÃO DE DESENVOLVIMENTO WEB ENTRE A TECNOLOGIA SERVLETS / JSP E O FRAMEWORK VRAPTOR

Trabalho de Conclusão de Curso pelo discente Augusto César Bevilaqua apresentado à Universidade Federal de Juiz de Fora como requisito para obtenção do diploma de conclusão do curso de Pós Graduação em Desenvolvimento de Sistemas com a Tecnologia Java.

ORIENTADOR: Prof. DAVES MARCIO SILVA MARTINS.

JUIZ DE FORA
2013

FOLHA DE APROVAÇÃO

BEVILAQUA, Augusto César, uma comparação de desenvolvimento Web entre a tecnologia Servlets/JSP e o framework VRaptor.

Trabalho de Conclusão de Curso apresentado pelo discente Augusto César Bevilaqua à Universidade Federal de Juiz de Fora - UFJF como requisito para obtenção do título de Pós Graduação em Desenvolvimento de Sistemas com a Tecnologia Java, realizado no 1º semestre de 2013.

BANCA EXAMINADORA

_____

MSC Daves Marcio Silva Martins
Professor orientador

_____

MSC Frederico Coelho Miranda
Professor(a) examinador(a)

_____

MSC Eduardo Pagani
Professor(a) examinador(a)

Examinado em: ____/____/_____

AGRADECIMENTOS

Agradeço a Deus pela valiosa oportunidade da vida, por me abençoar em cada passo e por essa conquista.

Aos meus queridos pais, Angélica e Edson, responsáveis pelo que sou hoje, pelo amor incondicional e apoio durante toda a vida, compartilhando alegrias e sonhos em todos os momentos, aos meus irmãos e demais membros da minha família, por preencherem meus dias com alegria, inclusive os que não estão mais presente entre nós, que também fazem parte desta conquista.

Ao meu orientador Daves Márcio da Silva Martins pela dedicação e conselhos.

A todos os mestres, pelos conhecimentos transmitidos no decorrer desses anos.

Aos verdadeiros amigos que conquistei nesse caminho, pela maravilhosa convivência, momentos de descontração e palavras de amizade que espero levar para sempre.

E, a todos que vibraram com a minha vitória, que apesar de não estarem aqui descritos, contribuíram para o sucesso de mais essa etapa, muito obrigado!

RESUMO

BEVILAQUA, Augusto César. **Uma comparação de desenvolvimento Web entre a Tecnologia Servlets/JSP e o Framework VRpator.** 51 f. Trabalho de Conclusão de Curso (Pós Graduação em desenvolvimento de sistemas com tecnologia Java). Universidade Federal de Juiz de Fora, 2013.

O desenvolvimento de aplicações *Java* tem se tornado cada dia mais interessante devido a várias qualidades da plataforma, como portabilidade e bibliotecas prontas que auxiliam no desenvolvimento e, a *API* (*Application Programming Interfaces*) que é uma biblioteca padrão *Java*.

Com a crescente demanda para o desenvolvimento web, a tecnologia *Java* teve que se adaptar a esse ambiente de desenvolvimento, nascendo assim os *Servlets/Jsp*. Com o tempo, essa tecnologia passou a não ser suficiente e foram necessários aprimoramentos. Assim começaram a surgir *Frameworks*, com proposta de desenvolvimento ágil e de fácil aprendizado.

Um desses *frameworks* é o *VRaptor*, que é um *Framework* MVC (*Model-View-Controller*) desenvolvido no Brasil e tem o foco no desenvolvimento ágil e rápido, onde une várias ferramentas em um único local.

O intuito, é o de comparar as tecnologias criando uma versão da mesma ferramenta de cadastro de clientes e suas funcionalidades utilizando o *VRaptor*, adaptam-se partes do código já feito, contudo, outras partes do código são excluídas e ou inseridas novas funcionalidades a fim de facilitar o desenvolvimento levando a um mesmo resultado final da aplicação.

A grande facilidade de utilização do *VRaptor* é a não necessidade de instalação de nenhum conteúdo adicional para a sua execução, ou seja, o profissional com pratica em desenvolver *JavaEE* com *Servlets/Jsp* não encontrará dificuldades, pois o desenvolvedor já trabalhava com esse ambiente.

Concluiu-se com este trabalho as vantagens e desvantagens no uso das duas ferramentas para o desenvolvimento *web*. Onde compara o início do desenvolvimento web até um dos *Frameworks* nacionais mais bem conceituados no desenvolvimento *Java*.

Palavras-chave: Java. VRaptor. Servlets. JSP. Framework. Desenvolvimento Web.

ABSTRACT

The Java application development has become increasingly important due to various qualities of the platform, such as portability and libraries ready to assist in the development , and the API (Application Programming Interface) which is a standard Java library .

With increasing demand for web development, Java technology had to adapt to this development environment , emerging as Servlets/Jsp. Over time , this technology has not been sufficient and necessary improvements. Thus began to emerge Frameworks with proposal development agile and easy to learn.

One of these frameworks is VRaptor, which is a Framework MVC (Model-View - Controller) developed in Brazil and has focused on developing agile and fast, which unites several tools in one place .

The aim is to compare the technologies creating a version of the same tool customer base and its functionality using VRaptor, adapt parts of the code already done, however, other parts of the code are inserted or deleted, and new features to to facilitate the development leading to the same final result of application.

The ease of use VRaptor is no need to install any additional content for their execution, the professional practice in developing with JavaEE Servlets/Jsp not find difficulties because the developer already working with this environment.

The conclusion of this work the advantages and disadvantages in the use of two tools for web development. Which compares the beginning of web development by one of the most reputable national frameworks in Java development .

Keywords: Java. VRaptor. Servlets. JSP. Framework. Web Development.

SUMÁRIO

O perfeito equilíbrio entre corpo e mente é aquela qualidade do homem civilizado que não apenas lhe dá superioridade em relação ao selvagem ou ao reino animal, mas também lhe fornece todos os poderes físicos e mentais que são indispensáveis para o alcance do objetivo do ser humano – saúde e felicidade.

Joseph Hubertus Pilates (1880-1967).

1 Introdução

O desenvolvimento de aplicações *Java* tem se tornado cada dia mais interessante devido a várias qualidades da plataforma, como portabilidade e bibliotecas prontas que auxiliam no desenvolvimento, como as API (*Application Programming Interfaces*) que é uma biblioteca padrão *Java*.

Com o foco no desenvolvimento web na plataforma *Java*, em 1997, surgem os *Servlets* (Caelum, 2012), que é uma ótima alternativa de interatividade do banco de dados com o cliente, gerando dinamicamente documentos *html* pré-definidos que qualquer navegador web (*Internet Explorer, Mozilla Firefox, Google Chrome* entre outros) consiga exibir e armazenar dados nas sessões. Apesar da última versão lançada ser a 3.0 em Dezembro de 2009 juntamente com o *Java EE 6*, as versões 2.x (mais precisamente a 2.4 e 2.5) são ainda as mais utilizadas pelo mercado (Caelum, 2012).

O *VRaptor* é utilizado para facilitar o desenvolvimento *web*, evitando contato com as classes pouco amigáveis do *javax.servlet*, deixando o código legível e desacoplado, facilitando a visualização.

Será visto que tanto *Servlets* como a *Jsp* funcionam em conjunto, onde cada um tem sua responsabilidade bem definida como, por exemplo, a parte de controle da aplicação aos *Servlets* e a parte de apresentação (páginas *web*) às páginas *Jsp*. Contudo, a diferença principal do *VRaptor* é a utilização dos *Controllers* como o centro de funcionamento da aplicação e explorando os *frameworks* disponíveis para a camada de visão.

O *VRaptor* consegue unir vários *frameworks* que já são tendências no mercado em um só lugar, como *Spring Security, Hibernate* entre outros. Apesar de tais ferramentas poderem ser utilizadas separadamente, com a linguagem *Jsp/Servlets*, se tem no *VRaptor*, todas essas configurações pré-definidas, facilitando o trabalho do desenvolvedor.

A *Caelum* criou o *VRaptor* que é um *framework mvc*[1] que segue todas as boas práticas de programação como *Conversion Over Configuration (Coc),* Injeção de dependências (*DI)*, Inversão de Controle (*IOC*), Separação de Responsabilidade, entre outros, e o vem aprimorando (Caelum, 2012).

[1] MVC – Conceito (paradigma) de desenvolvimento e *design* que tenta separar uma aplicação em três partes distintas: *Model*, *View* e *Controller*, para uma melhor divisão das camadas da aplicação (Edson Gonçalves - 2007).

O presente trabalho apresenta-se dividido em nove capítulos.

O primeiro capítulo fornece uma breve introdução sobre o tema de estudo.

O segundo capítulo busca caracterizar um *framework* e abrange informações a respeito do *MVC* assim como o histórico de sua evolução.

O desenvolvimento *Web* e o funcionamento da aplicação são os objetos de discussão do terceiro e quarto capítulo, que descrevem a evolução de softwares através da internet e a visão do mercado.

O modelo da aplicação a ser utilizada como comparação foi realizada no quinto capítulo.

O desenvolvimento de uma aplicação em *Servlets/Jsp* foi abordado no sexto e no sétimo capítulos, foi refeita a aplicação utilizando a tecnologia do *VRaptor*.

No oitavo capítulo há uma comparação entre *Servlets/Jsp* e *VRaptor*.

Com base nas informações anteriores, as considerações finais do trabalho são debatidas no nono e último capítulo.

2 *Servlets/JSP* X *VRaptor*

Segundo H. M. *Deitel* (2008), o "*Oak*", nome inicial da plataforma *Java*, foi criado por *James Goslin* que traduzido significa "Carvalho" em homenagem à árvore que via de sua janela do escritório em que trabalhava. Obteve financiamento direto da *Sun Microsystems* que pretendia desenvolver um *software* capaz de suprir o mercado de consumo eletrônico e aplicar a tecnologia em vários dispositivos como televisores, telefones, etc.

Porém, desde a criação inicial do projeto no inicio dos anos 1990, o "pai" da tecnologia *Java*, Gosling, foi incumbido de adaptar o *Oak* também para a *internet.* Em janeiro 1995, com uma nova versão do *Oak*, a plataforma foi rebatizada para o nome *Java.* (JavaFree.org. Java, 2012).

A tecnologia *Java* tinha sido projetada para se mover através de redes de dispositivos heterogêneos, redes como *internet*. Agora, aplicações poderiam ser executadas dentro dos *Browsers* nos *Applets Java* e tudo seria disponibilizado pela Internet instantaneamente. Foi o estático *HTML* dos *Browsers* que promoveu a rápida disseminação da dinâmica tecnologia. (JavaFree.org. Java, 2012).

2.1 O *Servlets / JSP*

Muitos desenvolvedores iniciantes aprendem *Java* para *web* utilizando o *JavaServerPages (JSP)* e logo após que começam a conhecer os *Servlets*. Segundo Edson Gonçalves (2007), os *Servlets* são base do desenvolvimento de qualquer aplicação *Java* escrita pra *web*, seja no uso de *JSP* ou até mesmo com outros *frameworks* como o *Java Server Faces (JSF).*

Ainda segundo Edson Gonçalves (2007), *JSP (Java Server Pages)* é uma tecnologia utilizada no desenvolvimento de aplicações para *Web*, assim como o *ASP (Active Server Pages)* da *Microsoft,* porém o *Java* não se prende ao uso da tecnologia apenas nos sistemas operacionais da *Microsoft*.

Apesar do *JSP* caminhar em conjunto com os *Servlets,* estes não dependem do uso da *JSP* para serem funcionais. Porém, para melhor organização da visão (estática) da lógica (dinâmica), é altamente recomendado esse trabalho. (Edson Gonçalves, 2007)

Segundo Edson Gonçalves (2007), *Servlets* são classes *Java*, desenvolvidas de acordo com uma estrutura definida que quando executadas podem tratar requisições recebidas de clientes.

Cada *Servlet* é um objeto *java* que recebe as requisições (*request*) do cliente e retorna envia uma resposta (*response*). Como exemplo, se pode citar: uma nova página *html,* um arquivo, uma imagem ou, inclusive, todos juntos.

2.2 O *VRaptor*

O *VRaptor3* é uma ferramenta *MVC* (*Model-View-Controller*) em *Java* focado no desenvolvimento rápido e simples, e na fácil manutenção do código. Usando muitas ideias e boas práticas que surgiram nos últimos anos, como Convenção sobre Configuração, Injeção de Dependências e um modelo *REST*, é possível fazer uma aplicação de uma maneira bastante agradável e produtiva. É também uma iniciativa brasileira, nascida dentro da Universidade de São Paulo usada em muitas empresas (Caelum, 2012).

Criado em 2003 no IME-USP, teve sua versão 2.0 lançada em 2005 e a versão 3.0 em 2009 e em constante atualização, (Softwarelivre.org, 2012).

2.3 A Escolha das Tecnologias

No passado, *Servlets/JSP* era uma das principais tecnologias usadas no desenvolvimento de aplicativo *web*, assim como *ASP* e *PHP*. Com a evolução das tecnologias, como o *ASP.NET* ou com novas versões do *PHP* por exemplo, o conceito de *Servlets/JSP* estava se tornando obsoleto, sendo necessário uma nova ideia que adicionaria recursos atuais e facilitaria o desenvolvedor em suas tarefas. Kurniawan (2002).

A tecnologia *Servlet* é base do desenvolvimento de aplicativos *web* usando a linguagem de programação *Java*. Ela é uma das tecnologias *Java* mais importante, e esta é

subjacente para outra tecnologia *Java*, popular para desenvolvimento de aplicativo: *Java Server Pages (JSP)*. Portanto entender a aplicação *Servlet* e sua arquitetura é importante. Ainda que o profissional queira apenas desenvolver seu aplicativo Java usando páginas *JSP*, entender a tecnologia *Servlet* ajuda a montar um aplicativo *JSP* mais eficiente e efetivo, pois esta página será compilada em *Servlets* pelo servidor. Kurniawan (2002).

Com essa evolução, a comunidade *Java* recebeu também novas ferramentas para o desenvolvimento. Começaram a surgir as primeiras versões dos *Frameworks JSF*, *Hibernate*, *Spring Security* e etc. O *VRaptor* surge então agrupando em uma única ferramenta o que de melhor tem as tecnologias oferecidas no mercado (*Hibernate*, *Spring*, etc), trazendo alta produtividade e facilidades dos mundos de *Rails* e *Grails* por exemplo, encapsulando a lógica dos *Servlets*. (AGOSTI, 2013).

Neste contexto, trabalharemos com o início do desenvolvimento *web* com o *Servlets/JSP*, comparando o seu resultado final no mesmo projeto utilizando as ferramentas atuais para o desenvolvimento *Java* para *web*, com o *Framework VRaptor*, analisando em ambos os casos, a lógica de programação, tamanho de código fonte, complexidade no desenvolvimento, dentre outros.

2.4 O que é *Framework*

"Framework é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação." - FAYAD e SCHMIDT (MOTTA, 2012).

Um *framework* é uma abstração que une códigos comuns entre vários projetos de *software*, provendo uma funcionalidade genérica. (AGOSTI, 2013).

O principal propósito de um *framework* é ajudar no processo de desenvolvimento de aplicações. Ele permite que as aplicações sejam desenvolvidas mais rapidamente e mais facilmente, e deve resultar em uma aplicação de qualidade superior. (MOTTA, 2012) .

Vários *frameworks* podem ser usados na construção de um único aplicativo de *software*. Para facilitar a escolha e o uso de vários *frameworks* durante o desenvolvimento de *software*, e garantir a integração, evolução e manutenção dos mesmos foi criada uma estrutura chamada *Framework* Integrador, que visa aumentar a padronização e a produtividade no desenvolvimento de *software*. (MOTTA, 2012).

2.4.1 O que é *MVC*

Edson Gonçalves (2007), cita que *MVC* é um conceito (paradigma) de desenvolvimento e *design* que tenta separar uma aplicação em três partes distintas. As três partes, se dividem em *Model, View e Controller* originando assim a abreviação *MVC* que são as três primeiras letras de cada divisão.

Model (modelo): responsável pelos dados do programa e as suas transformações, ou seja, trabalha com a parte de armazenamento e busca de dados. No caso de um cadastro de fornecedor, seria sua modelagem criando maneiras de inserir um novo fornecedor, alterar ou até mesmo excluir.

View (visão): apresentação visual dos dados processados pelo *Model* ao usuário. Um exemplo, seria a visão de um *menu* específico para cada usuário ao logar no sistema.

Controller (controlador): corresponde aos comandos enviados pelo usuário sobre os dados apresentados no modelo, levando-o a alterar conforme tais solicitações e a *view* que deverá ser exibida. Um exemplo desse trabalho seria o envio do pedido do usuário (ao *Controller*) para exibir uma listagem de usuários onde o *Model* busca as informações e a view as exibe.

Como boa prática de programação, não se deve ‘misturar’ as camadas, facilitando assim a manutenção do código além da segurança do código, sobretudo na *view* que é a que o usuário tem o contato direto com o sistema.

2.5 Vantagens da Tecnologia Servlets / JSP e o Framework VRaptor

O *VRaptor* foca em simplicidade e, portanto, todas as funcionalidades que serão vistas têm como meta resolver o problema do programador da maneira menos intrusiva possível em seu código.

Tanto para salvar, remover, buscar e atualizar ou ainda funcionalidades que costumam ser mais complexas como upload e download de arquivos, resultados em formatos diferentes (*XML*, *JSON*, *XHTML*, etc), tudo isso é feito por meio de funcionalidades simples do *VRaptor*, que sempre procuram encapsular *HttpServletRequest*, *Response*, *Session* e toda a *API* do *javax.servlet*. (Caelum, 2013).

Com o rápido aprendizado, não há necessidade de se preocupar com treinamento da equipe. É muito simples trabalhar com o *VRaptor*. Dois dias são suficientes para conhecer todo seu funcionamento. Nada de *HttpServletRequest*, *HttpServletResponse* e *HttpSession* (MADEIRA, 2012).

A ferramenta retira do desenvolvedor a responsabilidade de lidar diretamente com as classes da API dos Servlets através de simples anotações. (MADEIRA, 2012).

Com poucas configurações, o *VRaptor* trabalha por convenções o que diminui relativamente o número de configurações. Por exemplo, marca-se uma classe com a anotação *@Component*, automaticamente todos os métodos públicos estarão disponíveis para serem acessados pela view. (MADEIRA, 2012).

A convenção também é aplicada nos redirecionamentos após a execução de um método na *Action (Logic)*. O método não retorna nenhuma string ou objeto indicando para qual recurso a requisição deverá ser direcionada. O *VRaptor* irá direcionar para uma página *jsp* com o padrão '/nomeComponente/nomeMetodo.ok.jsp'. (MADEIRA, 2012).

Vale ressaltar que se for necessário é possível customizar o funcionamento de uma forma bem simples.

Um controlador mvc não deveria interferir na view. Sua responsabilidade é apenas controlar as requisições, direcionando-as para as classes responsáveis por tratá-las. Encher a JSP com taglibs de frameworks MVC é muito perigoso e arriscado. Um padrão (JSTL) é mais adequado, e caso seja necessário crie as suas próprias. (MADEIRA, 2012).

3 O Desenvolvimento *Web*

No início do desenvolvimento de uma nova aplicação, há sempre o questionamento de qual a melhor tecnologia que se deve buscar? Qual linguagem a ser utilizada ou o banco de dados? Porém, e não muito menos importante, faltou colocar em foco se será uma aplicação com linguagem voltada à *web* ou para *desktop*.

A grande vantagem que se pode citar no desenvolvimento *web* é o suporte centralizado, no qual em uma possível manutenção, não há necessidade de aplicá-lo em todos os 'terminais' que utilizam esse acesso, e sim somente em um único lugar, ou seja, no servidor web que gerencia a aplicação.

O Desenvolvimento de softwares pela *web* conta com simplicidade. O usuário hoje tem contato mais freqüente com a infraestrutura de acesso à *internet* (*e-mails*, páginas de notícias, etc). Porém, não basta ser um sistema voltado à *internet*, deve ser de fácil utilização para o cotidiano do usuário e que o atenda em suas necessidades.

Apesar de grande parte de aplicações *web* funcionarem no servidor, para o bom funcionamento da estrutura, os aplicativos dependem dos recursos também da máquina do usuário e seus navegadores de *internet* (*browsers*) instalados, além de alguns recursos adicionais como a *Java Runtime*, por exemplo, para executar aplicações *Java*.

A instabilidade da *Internet* do usuário também prejudica o uso da aplicação *web*, pois atrapalha a utilização no máximo da ferramenta elaborada, podendo acarretar erros de visualização e em muitos casos até mesmo no processamento das informações.

Atualmente os servidores *web* estão ganhando novas capacidades que os permitem executar programas, acessar bancos de dados corporativos e se comunicarem com outros objetos da rede ficando mais estáveis, assim como a conexão à *internet* tanto pelas empresas quanto pelos clientes, minimizando deste modo, os problemas.

4 O Funcionamento da Aplicação *Web*

O destino final de uma aplicação web, é sempre a tela do cliente, através de um navegador web (*Internet Explorer, Mozilla Firefox, Google Chrome*, entre outros), ele acessa diretamente um servidor em qualquer localidade do mundo que contém a lógica da aplicação, seja ela um simples código *html*, uma imagem, ou até mesmo uma aplicação complexa em *Java*.

No caso básico de uma aplicação *web*, acessa-se o servidor em busca de uma lógica qualquer (consulta a clientes, por exemplo), e como retorno tem-se, por exemplo, uma página *html* para que o usuário tenha essa visão final como apresentado na Figura 1. Tal visão, nada mais é que a resposta do servidor. Não necessariamente, poderia ser apenas *um html, php, ou asp*, mas um *pdf, flash*, imagem entre outros.

Assim, determina-se que a aplicação, possui duas etapas de desenvolvimento: a primeira etapa é executada a lógica do sistema no servidor. E, na segunda parte, retorna-se o cliente renderizando o resultado da lógica para ser apresentada ao usuário.

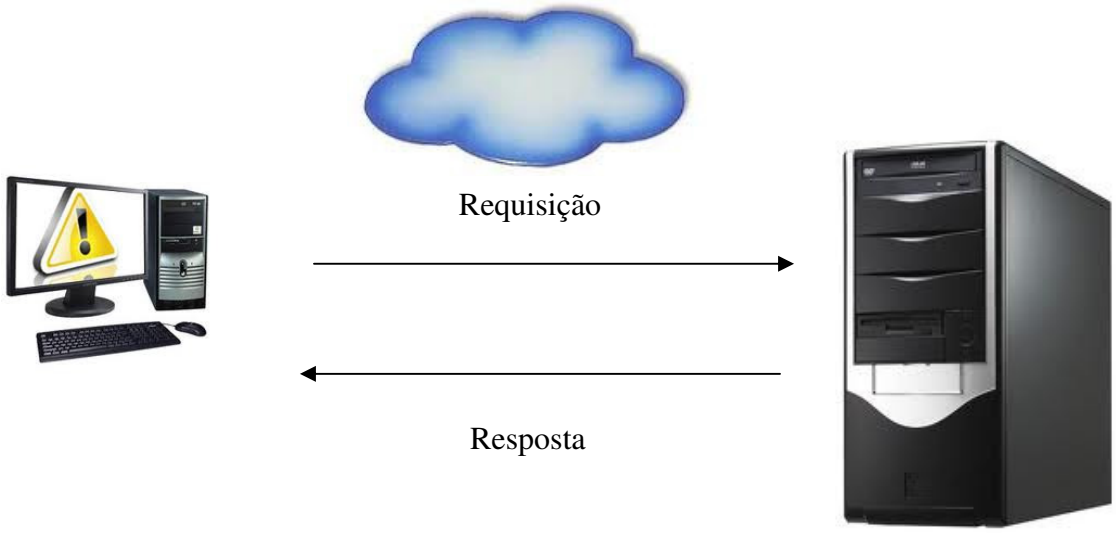

Figura 1. Requisição e Resposta

No *VRaptor*, a lógica de funcionamento funciona em torno do '*Controller*' que administra todas requisições

Primeiramente, tem-se um conjunto de lógicas, ou seja, um conjunto de operações que gostaríamos de executar, como por exemplo: AdicionarCliente, AlterarCliente, Excluir Cliente. Tal demonstração pode ser constatada na Figura 2.

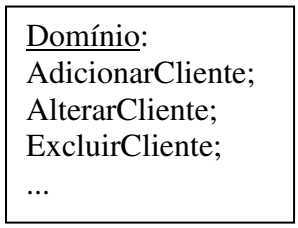

Figura 2. Conjunto de Operações

Posteriormente, tem-se a parte de visualização, ou seja, o que vai ser processado e mostrado ao cliente via browser. Podemos constatar na Figura 3

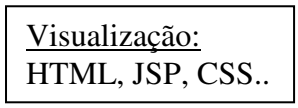

Figura 3. Visualização

Essas partes andam sempre juntas, pois sempre que se processa algo que o cliente solicite ao servidor, haverá um resultado final a ser visualizado. A Figura 4 apresenta o procedimento.

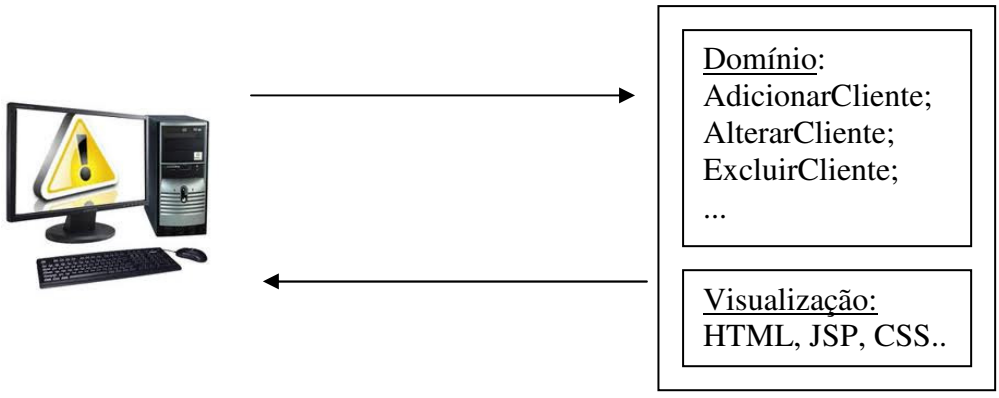

Figura 4. Resultado Final

Porém, nesse meio, existe a camada dos *Controllers*, na qual o *VRaptor* trabalha. Esta camada é responsável por controlar a nossa aplicação, retirando completamente a ideia de uso de *servlets* (*post, get*), no tratamento das requisições. Cabe aos *controlers* o acesso ao domínio e aos modelos de negocio que lá estão (AdicionarCliente, AlterarCliente, etc) ou '*model'*, assim como o acesso a visualização do resultado ou simplesmente '*views'*. A quinta Figura se baseia no exposto acima.

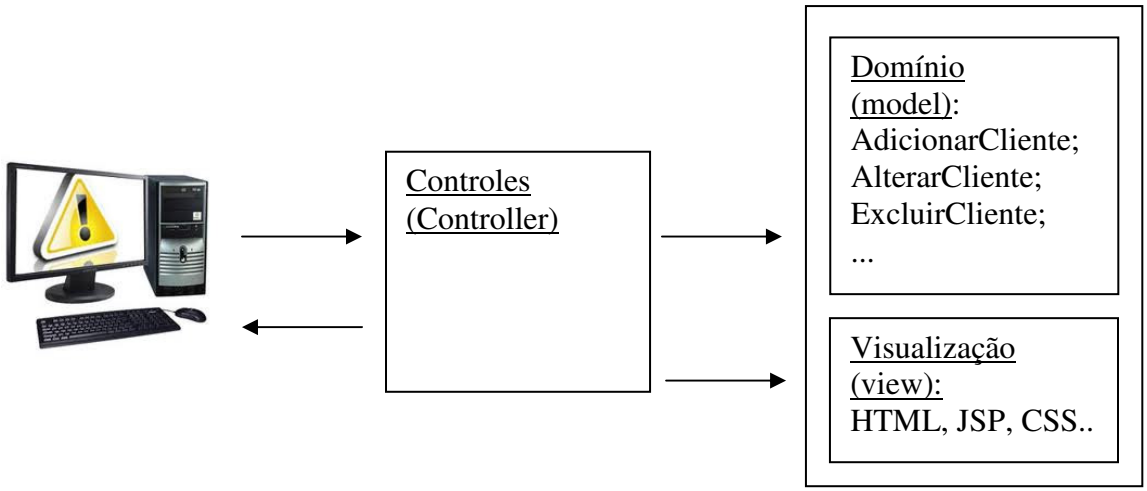

Figura 5. Visualização

Na figura 5 é exemplificado todo o processo, ou seja:

O cliente faz uma requisição à aplicação;

O *controller* acessa o modelo do domínio com as regra de negócio solicitada;

Com a resposta, o *controller* redireciona à *view* para a exibição do resultado obtido, ou até mesmo a tela de erro;

Com todas as informações necessárias, o *controller* retorna ao usuário final, a tela de resultado.

Importante: toda requisição da aplicação irá primeiramente passar pelo *controller*. Assim o *VRaptor* controlará toda a aplicação. Por esse motivo o *VRaptor* é chamado de '*FrontController'*, ou seja, é primeiramente por ele que passam as requisições.

Com essa divisão da parte de domínio e visualização, o código *Java* fica separado dos códigos de visualização, ou seja, os códigos *Java* permanecem no *model*, enquanto a visualização (*html, css, jsp, flash, javascript*, etc) se encontra apenas na *view*.

4.1 O Mercado para *JSP/Servlets* e *VRaptor*

O *Java,* hoje pertencente à *Oracle*, encontra-se no mercado desde 1995, com sua plataforma definida, concreta e disponibilizada gratuitamente a todos interessados. De outro lado, temos um *framework* que trabalha com a mesma tecnologia, porém com 'maneiras' de facilitar esse desenvolvimento, cabendo ao analista se adequar com a premissa de facilidade e desempenho futuro. (Oracle, 2013).

Ambas as tecnologias, têm a mesma premissa, ou seja, no final uma aplicação web em *Java* é gerada, porém, por caminhos diferentes. Inclusive tais tecnologias são executadas no mesmo servidor de aplicação '*Tomcat'*, onde nem é necessário nenhum adicional para executar projetos das duas tecnologias em paralelo.

O principal pilar do desenvolvimento *Web* em *Java* é a *API* de *Servlets*. Com ela é possível executar código de uma determinada classe *Java* a partir de requisições *HTTP* para uma *URL*. Esse acesso é feito através de configurações não triviais de um servidor e arquivos específicos de maneira diferente às antigas linguagens de *script*, como *Perl*, o que pode tornar o aprendizado de *Servlets* complicado.

Enquanto nos *Servlets* faz-se toda a parte de 'chamada' (*HttpServletRequest*) e 'resposta' (*HttpServletResponse*), assim como sessões (*HttpServletSession*), no *VRaptor* não se tem essa preocupação pois ele mesmo faz o tratamento pelas *API* dos *Servlets* por simples anotações no código.

Até a versão 2 da *API* de *Servlets*, precisa-se fazer a declaração dessas classes através de *XML* (o `web.xml`).

Um exemplo prático de uma definição de um *Servlet* em uma aplicação pela *API* pela versão dois seria o exemplo da imagem 1:

```
Web.xml
<servlet>
      <servlet-name>GeraSenha</servlet-name>
      <servlet-class>br.com.projeto.GeraSenha</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>GeraSenha</servlet-name>
      <url-pattern>/GeraSenha</url-pattern>
</servlet-mapping>
```

Figura 6. Exemplo prático de uma definição de um *Servlet*

O *VRaptor* utiliza a versão 3.0 da *API* de *Servlets*, a partir da versão 3.1.

A última versão da *API* de *Servlets*, a 3.0, lançada em Dezembro de 2009 com o *Java EE 6*, traz algumas facilidades em relação às configurações das *Servlets*. De modo geral, não é mais preciso configurar no *web.xml* sendo suficiente usar a anotação *@WebServlet* apenas.

```
@WebServlet(value="/oiMundo")
public class OiMundo extends HttpServlet {
        PrintWriter out = response.getWriter();
```

Figura 7. Anotação do *WebServlet*

Pode-se acessar nossa *Servlet* através de uma *URL* semelhante a `http://localhost:8080/projeto/OiMundo?nome=MeuNome`, porém é mais interessante ter um formulário *HTML* que enviará a requisição e o parâmetro para a *Servlet*. Pode-se criar um `oiMundo.html` com o seguinte conteúdo:

```
<html>
  <body>
    <form action="oiMundo">Informe o nome:
      <input name="nome" type="text" />
      <input type="submit" value="Enviar" />
    </form>
  </body>
</html>
```

Figura 8. Exemplo de página *jsp*

5 Requisitos da Aplicação Padrão Proposta

A aplicação proposta descreve a elaboração e concepção de um cadastro de clientes, que busca uma melhor maneira de armazenar tais informações para simplificar a localização dos seus dados bem como ajuste das informações inseridas, “aposentando” as fichas cadastrais até então utilizadas em papel, deixando em aberto futuras adequações e expansões ao projeto.

5.1 Diagrama Entidade Relacionamento

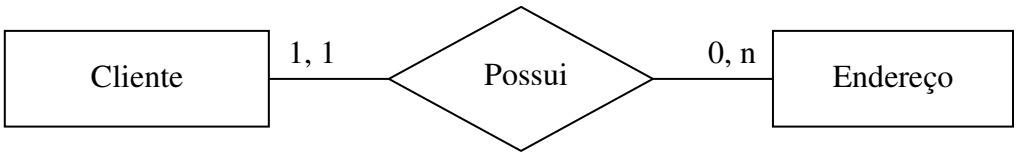

Figura 9. Diagrama Entidade Relacionamento (DER)

5.2 Caso de Uso

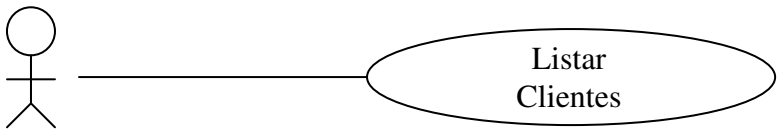

Figura 10. Caso de Uso da Aplicação

5.3 Banco de Dados

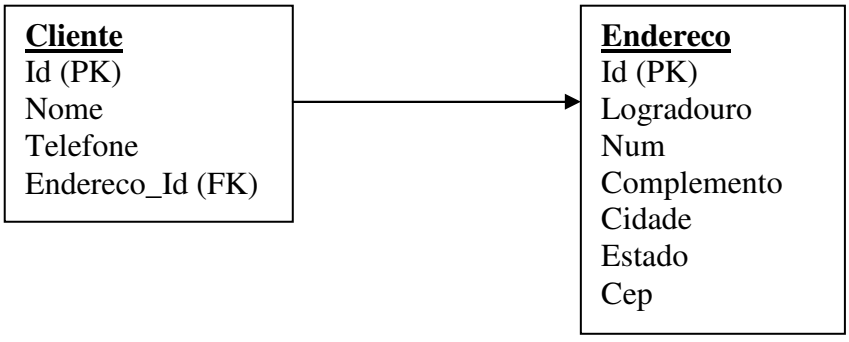

Figura 11. Estrutura do Banco de Dados

5.3.1 Script Geração Banco de Dados (*MySQL*)

```
CREATE TABLE `cliente`
(     `id` integer (11) NOT NULL AUTO_INCREMENT,
      `nome` varchar (255),
      `telefone` varchar (255),
      `endereco_id` integer (11),
      PRIMARY KEY (`id`)
) TYPE=InnoDB CHARACTER SET latin1 COLLATE latin1_swedish_ci;

ALTER TABLE `projetopos`.`cliente` ADD INDEX `FK96841DDAEA37681D`
(`endereco_id` );

CREATE TABLE `endereco`
(     `id` integer (11) NOT NULL AUTO_INCREMENT,
      `cep` varchar (255),
      `cidade` varchar (255),
      `complemento` varchar (255),
      `estado` varchar (255),
      `logradouro` varchar (255),
      `num` integer (11) NOT NULL,
      PRIMARY KEY (`id`)
) TYPE=InnoDB CHARACTER SET latin1 COLLATE latin1_swedish_ci;

ALTER TABLE `cliente` ADD CONSTRAINT `FK96841DDAEA37681D` FOREIGN KEY
(`endereco_id`) REFERENCES `endereco`(`id`);

SET FOREIGN_KEY_CHECKS=1;
```

Figura 12. Script de criação do Banco de Dados

6 Desenvolvendo uma aplicação padrão com *Jsp/Servlets*

Abaixo, será utilizado o desenvolvimento em *Jsp/Servlets* como primeiro exemplo comparativo das ferramentas. A estrutura final do projeto será como a exibida na imagem a seguir:

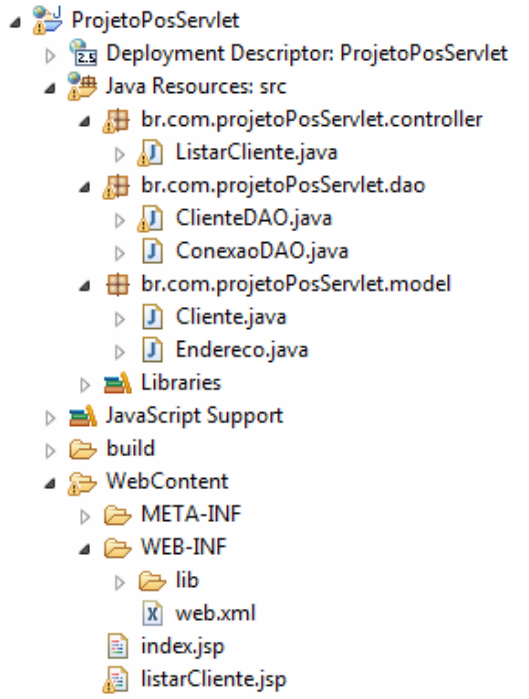

Figura 13. Estrutura final do projeto *Jsp/Servlets*

6.1 Criação

Apesar da facilidade inicial da criação de uma aplicação em *Jsp/Servlets*, posteriormente, o trabalho para a implementação da aplicação final será mais complicado. No *Eclipse JEE*, vá no caminho: *File* > *New* > *Dynamic Web Project*. Importante: É necessária a pré-configuração do *Apache Tomcat 6.0* no *Eclipse*.

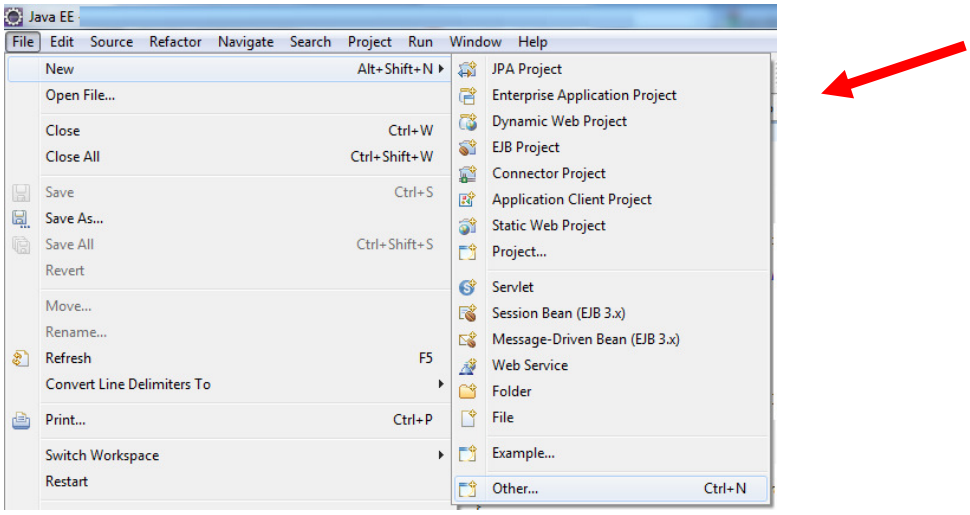

Figura 14. Pré-configuração do Apache no Eclipse

Escolha um nome para o Projeto e clique em "*Finish*":

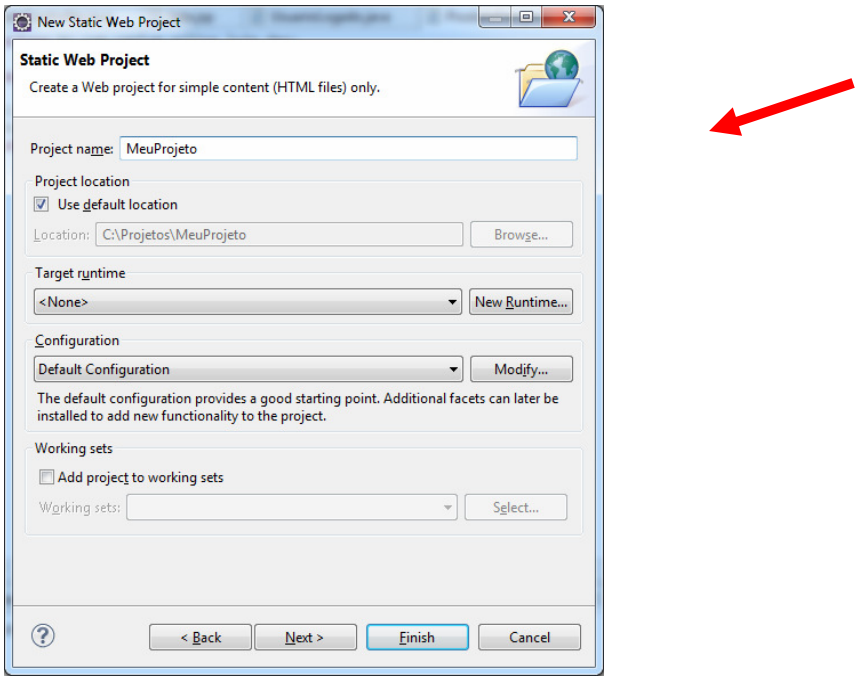

Figura 15. Criação do Projeto

6.2 A classe *Model* (Modelo)

É a classe que define todos os atributos de nossa classe e as variáveis que são necessárias em um determinado objeto.

Tal regra vale tanto para desenvolvimento em *Jsp/Servlets* como para o *VRaptor*.

Cria-se a seguinte classe: '*Cliente.java*' conforme abaixo, iniciando com as variáreis id, nome, telefone, endereço e, tornam-se mais necessárias, assim como criado para *Jsp/Servlets.:*

```
package br.com.projetoPosServlet.model;

public class Cliente {

    private int id;
    private String nome;
    private String telefone;
    private Endereco endereco;

   // getter e setters e métodos de negócio que julgar necessário
}
```

Figura 16. Classe Cliente - Servlet

Nota-se que esta classe possui somente as características do objeto 'Cliente' de modo que todos os objetos criados utilizarão tais parâmetros. Observa-se que existe também o objeto endereço que fará relacionamento com objeto 'Cliente' em sua criação. Desta forma, modela-se a classe '*Endereco.java*':

```
package br.com.projetoPosServlet.model;

public class Endereco {

    private int id;
    private String logradouro;
    private int num;
```

```
    private String complemento;
    private String cidade;
    private String estado;
    private String cep;

   // getter e setters e métodos de negócio que julgar necessário
```

Figura 17. Classe Endereço - Servlet

6.3 *Controller*, o coração da aplicação *JSP/Servlet*

Nos *Servlets / Jsp* é o pacote que acomoda as classes responsáveis pelas chamadas de requisição da aplicação. Por regra cria-se um pacote '*controller*' e nele inserem-se todos os *Servlets* que o aplicam.

Servlets são classes *Java*, que tratam requisições recebidas de clientes.

Tais classes podem, inclusive, fazer a chamada ao banco de dados, e retornar um resultado desse acesso (sucesso ou erro).

Cria-se um novo *Servlet* em *File* > *New* > *Servlet*. Nota-se que a classe será gerada já no formato necessário, e adicionada no *web.xml* a referencia necessária do *servlet*.

Web.xml

```
...
<servlet>
    <description></description>
    <display-name>ListarCliente</display-name>
    <servlet-name>ListarCliente</servlet-name>
    <servlet-
class>br.com.projetoPosServlet.controller.ListarCliente</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ListarCliente</servlet-name>
    <url-pattern>/ListarCliente</url-pattern>
  </servlet-mapping>
...
```

Figura 18. Alteração do web.xml

Faz-se as configurações do acesso ao banco de dados através da classe *DAO (Data Acess Object*), que contem as permissões de acesso ao banco de dados. Essa classe se chamará '*ConexaoDAO.java'* e está ajustada para a conexão do banco de dados *'MySQL'*, podendo ser adaptada à outros bancos disponíveis no mercado.

```
package br.com.projetoPosServlet.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexaoDAO {

	// Método Construtor//
	public static String status = "Não conectou...";
		public ConexaoDAO() {
	}

	// Método de Conexão//
	public static java.sql.Connection getConn() {
	  Connection connection = null;  //atributo do tipo Connection
		try {
			// Carregando o JDBC Driver padrão
			String driverName = "com.mysql.jdbc.Driver";
			Class.forName(driverName);
			// Configurando a conexão do banco de dados//
			String serverName = "localhost"; //caminho ou ip do BD
			String mydatabase ="projetopos"; //nome do BD
			String url = "jdbc:mysql://"+serverName+"/"+mydatabase;
			String username = "root";  //nome usuário do BD
			String password = "";      //senha de acesso ao BD

			connection = DriverManager.getConnection(url, username,
					password);
```

```
            return connection; // Retorna conexão com sucesso

            } catch (ClassNotFoundException e) {  //Driver não encontrado
                System.out.println("O driver nao foi localizado.");
                return null;
            } catch (SQLException e) { //Erro ao conectar no banco
                System.out.println("Nao conectou ao Banco de Dados.");
                return null;
            }
    }

    // retorna o status da conexão//
    public static String statusConection() {
      return status;
    }

    // fecha a conexão//
    public static boolean fecharConexao() {
      try {
            ConexaoDAO.getConn().close();
            return true;
      } catch (SQLException e) {
            return false;
      }
    }

    // Método que reinicia sua conexão//
    public static java.sql.Connection ReiniciarConexao() {
      fecharConexao();
      return ConexaoDAO.getConn();
    }
}
```

Figura 19. Classe ConexaoDAO de acesso ao banco de dados

É importante observar que para conexão a um banco de dados, é necessário um arquivo de conexão geralmente criado pelo próprio desenvolvedor do banco. No projeto, foi utilizado o arquivo de conexão do *MySQL* de nome '*mysql-connector-java-5.1.18-bin.jar*' e adicionado na pasta do projeto 'WebContent > WEB-INF > lib'.

Precisa-se também das classes *DAO (Data Acess Object*) para as buscas específicas no banco de dados. A classe que retornará os dados do cliente se chamará ClienteDAO e englobará todas as '*querys'* de acesso ao banco de dados. O método '*listar*()' será criado para retornar uma lista de clientes.

```
package br.com.projetoPosServlet.dao;

//imports necessários...

public class ClienteDAO {

	public static List<Cliente> listar(){
		List<Cliente> clientes = new ArrayList<Cliente>();
		ConexaoDAO conn = new ConexaoDAO();
		try{
			Statement stmt = (Statement) conn.getConn().createStatement();
			ResultSet rs = stmt.executeQuery
				("SELECT * FROM cliente, endereco " +
				 "WHERE endereco_id = endereco.id");
			while(rs.next()){
				Cliente cliente = new Cliente();
				cliente.setId(rs.getInt("id"));
				cliente.setNome(rs.getString("nome"));
			    Endereco endereco = new Endereco();

				endereco.setCidade(rs.getString("cidade"));
				endereco.setEstado(rs.getString("estado"));

				cliente.setEndereco(endereco);

				clientes.add(cliente);
			}

		}catch (Exception e) {
			e.printStackTrace();
		}finally{
			conn.fecharConexao();
		}
		return clientes;
	}
}
```

Figura 20. Classe ClienteDAO – dados do cliente

O *Servlet* que fará a chamada da lista quando o usuário clicar na opção, será o *'ListaCliente.java'*, e ficará da seguinte forma:

```
package br.com.projetoPosServlet.controller;

//imports necessários...

public class ListarCliente extends HttpServlet {
	private static final long serialVersionUID = 1L;

    public ListarCliente() {
        super();
    }

	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		Service(request, response);
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		Service(request, response);
	}

	protected void Service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		ClienteDAO clienteDao = new ClienteDAO();

		List<Cliente> clientes = ClienteDAO.listar();
		request.setAttribute("clientes", clientes);

		RequestDispatcher rd =
			request.getRequestDispatcher("listarCliente.jsp");
		rd.forward(request, response);
	}
}
```

Figura 21. Classe ListarCliente – retorna dados dos clientes

Após o processamento do *Servlet*, o resultado será direcionado à página "*listarCliente.jsp*", conforme a seguir:

```
<%@page import="br.com.projetoPosServlet.dao.ClienteDAO"%>
<%@page import="br.com.projetoPosServlet.model.Cliente"%>
<%@page import="java.util.List"%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Lista Clientes</title>
</head>
<body>
	<h3>Relação de Clientes</h3>
	<table border=1>
		<tr>
			<td width="300px">Nome</td>
			<td width="150px">Cidade</td>
			<td width="20px">Estado</td>
		</tr>
		<%
		List<Cliente> clientes =
				(List<Cliente>)request.getAttribute("clientes
			");
		for (Cliente c : clientes){ %>
		<tr>
			<td><%=c.getNome()%></td>
			<td><%=c.getEndereco().getCidade()%></td>
			<td><%=c.getEndereco().getEstado()%></td>
		<%}%>
		</tr>
	</table>
</body>
</html>
```

Figura 22. Página listarCliente.jsp - exibe dados dos clientes

Ao executar a aplicação, a mesma será exibida no *browser* e finalmente poderá ser visualizada em tela todos os Clientes e seu Endereço (cidade e estado) cadastrados no Banco de Dados da aplicação, como imagem abaixo:

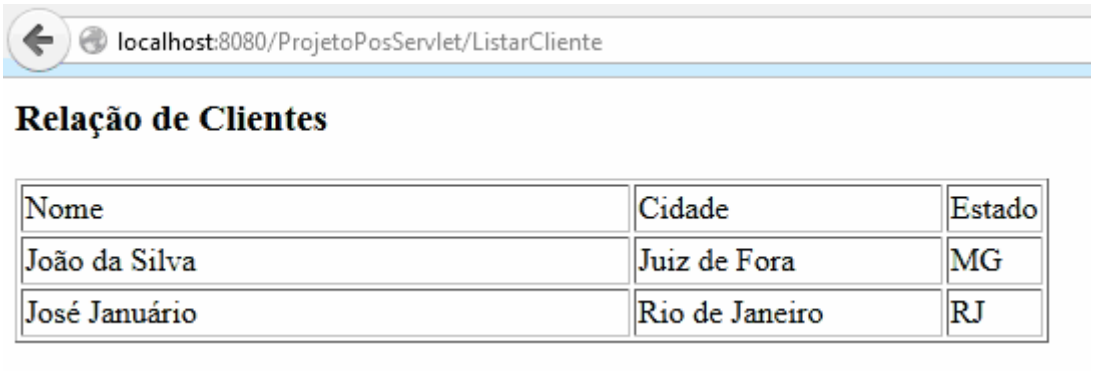

Figura 23. Visualização do projeto *Jsp/Servlets* em execução

7 Desenvolvendo uma aplicação padrão no *VRaptor*

Com intuito de comparar as tecnologias cria-se uma versão da mesma ferramenta de cadastro de clientes e suas funcionalidades utilizando o *VRaptor*. Pode-se adaptar parte do código já feito, porém serão excluídas partes do código original e ou inseridas novas funcionalidades no intuito de facilitar o desenvolvimento, levando a um mesmo resultado final da aplicação.

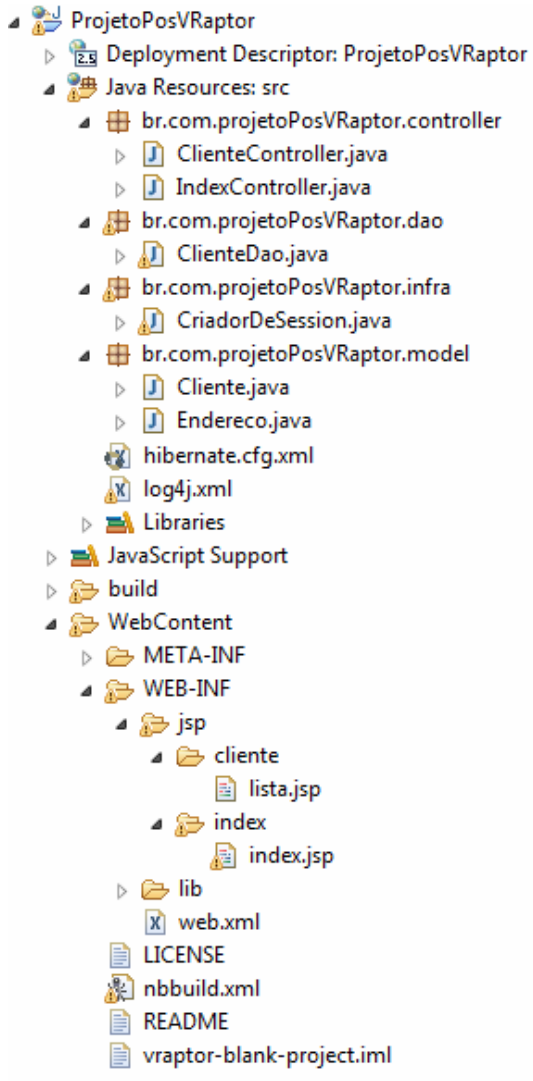

Figura 24. Estrutura final do projeto *VRaptor*

7.1 Download do *framework*

O *VRaptor*, para o seu funcionamento, precisa de algumas bibliotecas que acompanham o projeto '*blank*', que possui o conteúdo padrão de desenvolvimento dessa ferramenta. É importante salientar que é necessária a pré-configuração do *Apache Tomcat 6.0* no *Eclipse* para a execução do *VRaptor*.

Na página http://vraptor.caelum.com.br/download.jsp, baixar o conteúdo do projeto '*blank*'. Geralmente ele vem como um arquivo *zip*. O conteúdo deste exemplo foi baixado pelo link: http://vraptor3.googlecode.com/files/vraptor-blank-project-3.4.0.zip . Veja a seguir o local:

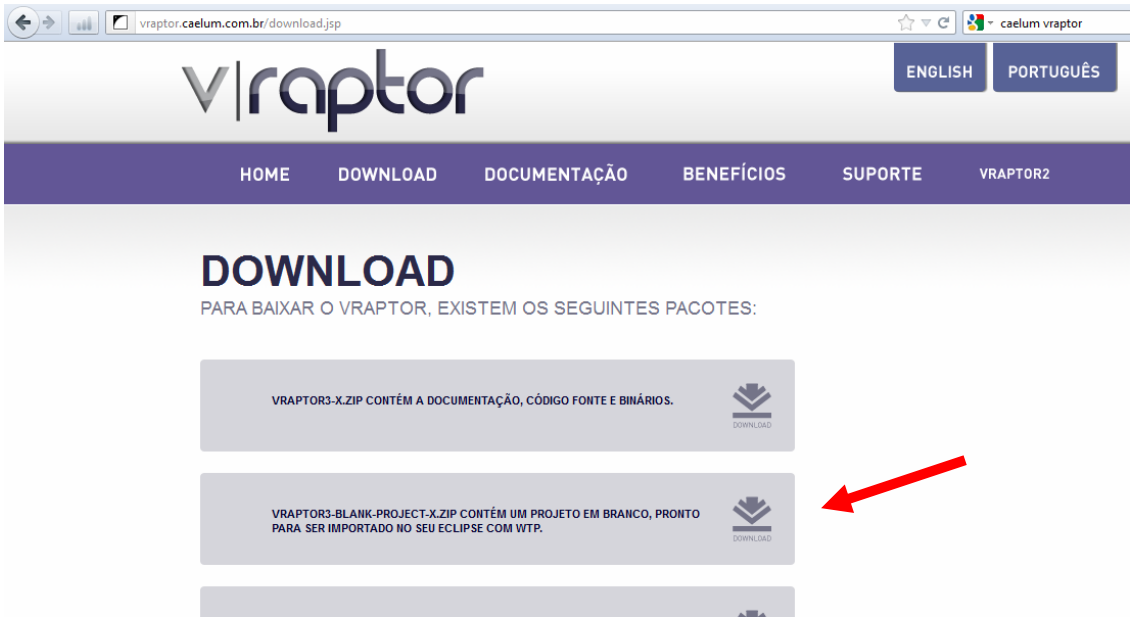

Figura 25. Pacotes

Após o *download*, descompactar o arquivo em alguma pasta de sua preferência. Veja abaixo os arquivos já descompactados:

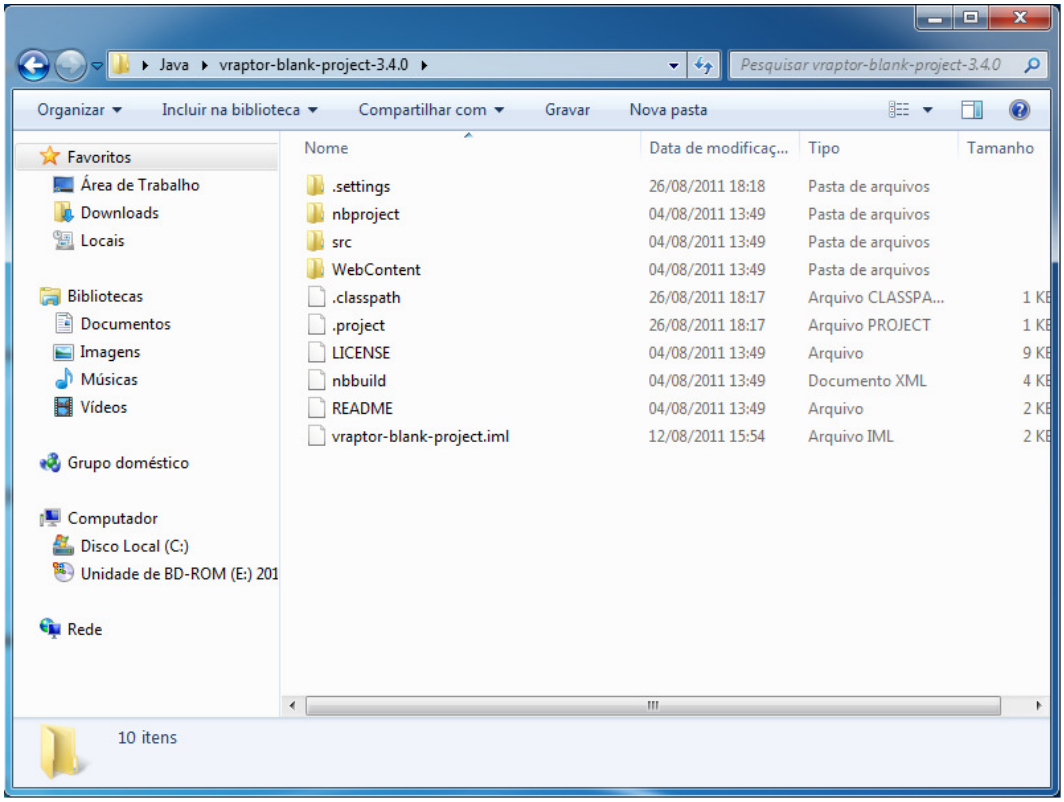

Figura 26. Demonstração

7.2 Importação do projeto

A grande facilidade de utilização do *VRaptor* é a não necessidade de instalação de nenhum conteúdo adicional para a sua execução. Ou seja, quem já era acostumado a desenvolver *JavaEE* com *Servlets/Jsp*, não irá encontrar dificuldades, pois o desenvolvedor já trabalhava com esse ambiente.

No *menu* principal do *Eclipse*, vá em '*File* > *Import*", depois, clique na pasta '*General* > *Existings Projects into Workspace*" e clique em "*Next* >"

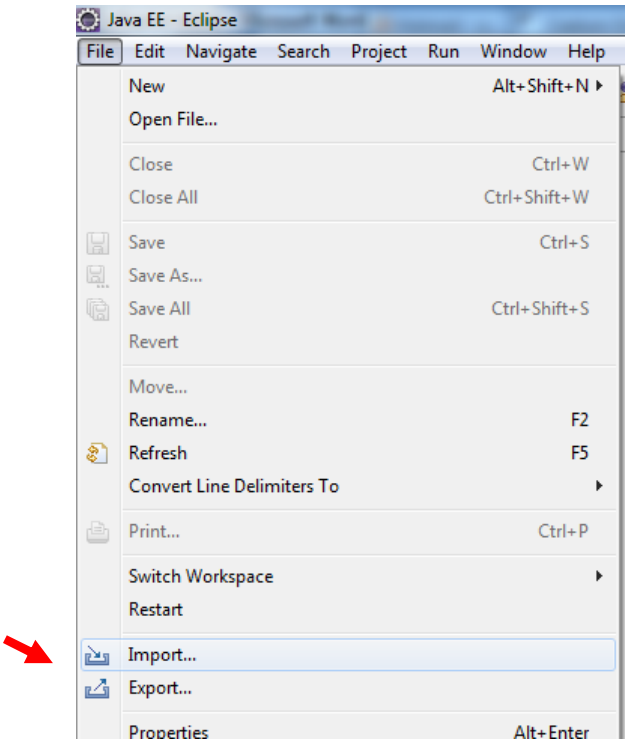

Figura 27. Tela de visualização

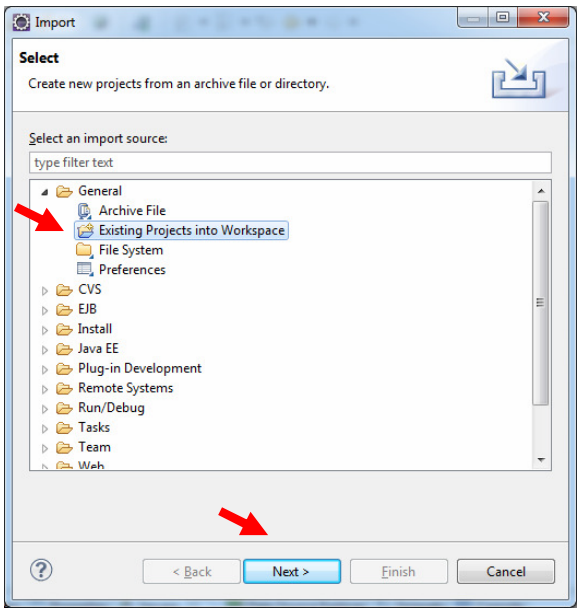

Figura 28. Tela de visualização

Clique em *Browse* e localize onde descompactou o projeto, deixe marcada a opção '*Copy projects into workspace'*, e depois clique em "*Finish*":

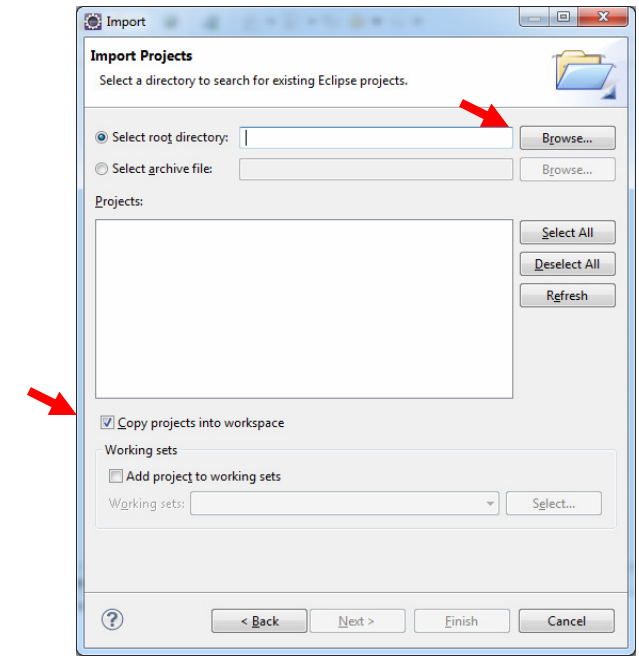

Figura 29. Demonstração

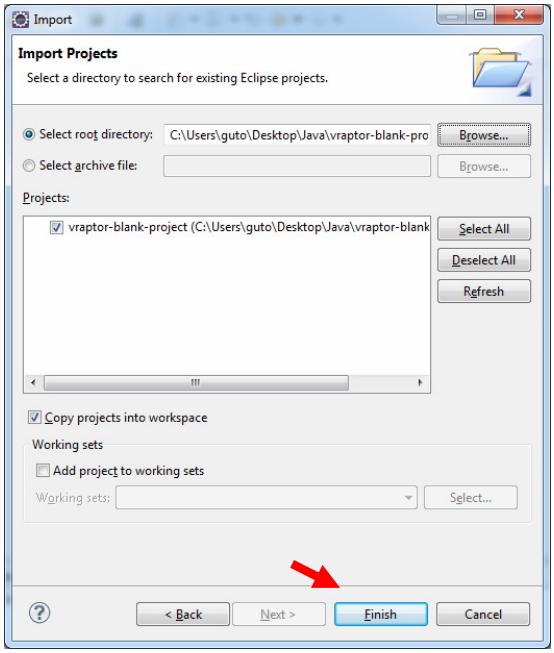

Figura 30. Demonstração

7.3 Teste de execução do *VRaptor*

Com o projeto já importado, clique com o botão direito sobre ele, '*Run* as > *Run* as *server*" . Como abaixo:

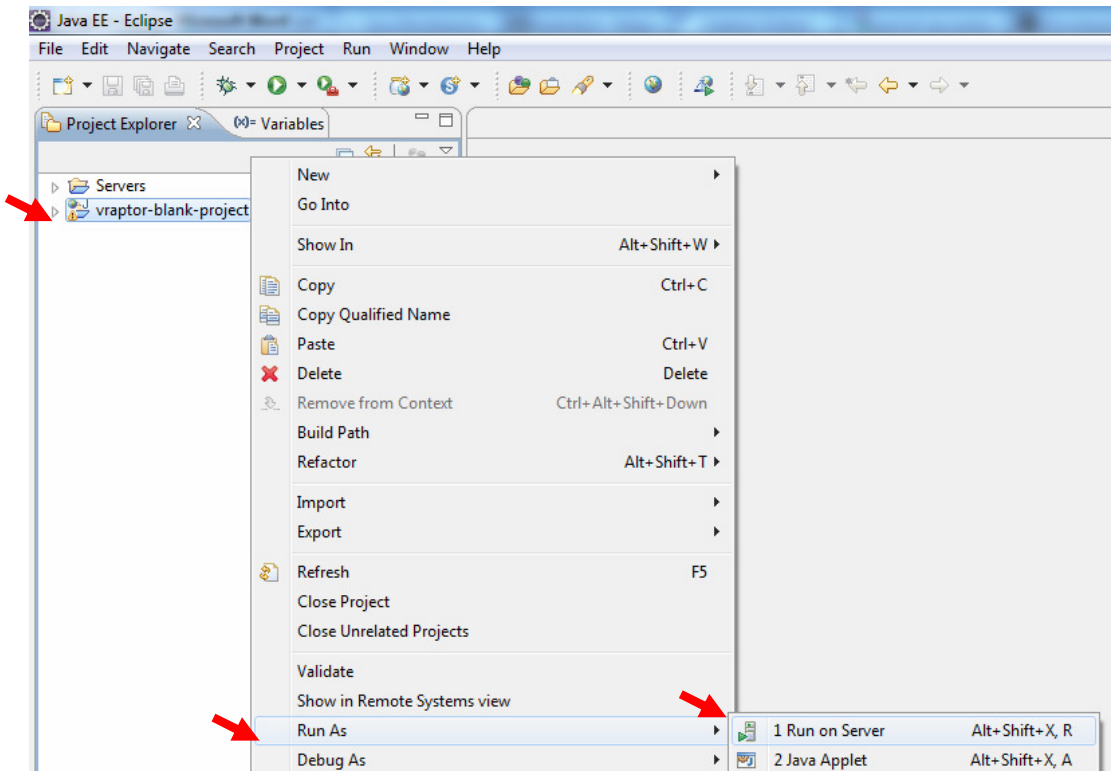

Figura 31. Visualização

Nota-se que no console aparecerá as bibliotecas do *VRaptor* já carregando...

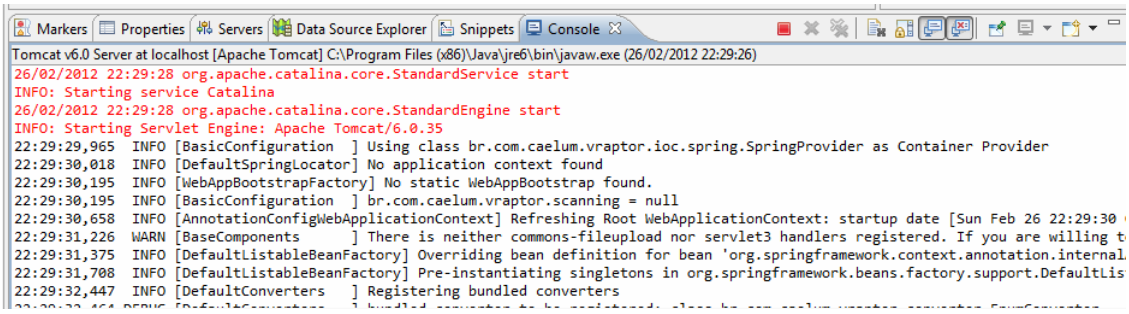

Figura 32. Bibliotecas carregando

Se for carregada a mensagem no *browser*: "*It works*!! *VRaptor*! */vraptor-blank-project/*", o primeiro projeto '*blank*' do *VRaptor* funcionará com sucesso e já estará apto a iniciar o projeto. Veja a seguir:

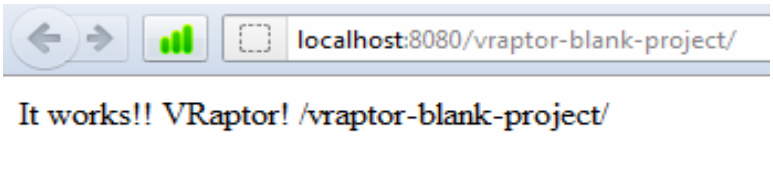

Figura 33. Primeiro Projeto

7.4 A classe *Model* (Modelo)

É a classe que define todos os atributos da classe criada e as variáveis que precisarão em um determinado objeto.

Tal regra vale tanto para desenvolvimento em *Servlets/Jsp* como para o *VRaptor*.

Cria-se a seguinte classe '*Cliente.java*' conforme abaixo obtendo as variáreis, id, nome, telefone, endereço e quantos mais tornam-se necessários. Assim como criado para *Jsp/Servlets*:

```
package br.com.projetoPosVRaptor.model;

public class Cliente {

      private int id;
      private String nome;
      private String telefone;
      private Endereco endereco;

    // getter e setters e métodos de negócio que julgar necessário
}
```

Figura 34. Classe Cliente - *VRaptor*

Para o *VRaptor*, é necessário colocar algumas anotações, de modo a facilitar o seu uso na aplicação, como a seguir:

```java
package br.com.projetoPosVRaptor.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
public class Cliente {

	@Id
	@GeneratedValue
	private int id;
	private String nome;
	private String telefone;

	@ManyToOne
	@JoinColumn(name="endereco_id")
	private Endereco endereco;

    // getter e setters e métodos de negócio que julgar necessário
}
```

Figura 35. Classe Cliente com anotações – *VRaptor*

Neste ponto, utilizam-se algumas anotações do JSR-220, a famosa *JPA*, na qual o *VRaptor* também tem esse suporte para que a própria aplicação faça a criação automática das tabelas necessárias para criação na base de dados.

O mapeamento via *annotations* dispensa o uso do arquivo '*xml*' de configuração para tal e passamos totalmente às declarações para a própria classe *Java* responsável em modelar o objeto.

Utilizando a funcionalidade no *Hibernate*, o *VRaptor* emprega as anotações neste primeiro momento. São muito convenientes para a criação do banco de dados, onde o desenvolvedor não precisaria se preocupar em modelar a estrutura do mesmo.

Além de aplicar a anotação '@*Entity*' informando que esta é uma entidade, tem-se a anotação '@*Id*' para determinar a chave primária no caso da criação do banco de dados e que

a mesma é auto incremental, ou seja, o banco de dados se encarrega de incrementar o próximo número de registro com a anotação '*@GeneratedValue*'.

Observa-se que a parte mais importante é quanto à utilização das chaves estrangeiras de maneira simples no código onde aproveitam-se a anotação *@ManyToOne* informando que a variável será uma chave estrangeira, e o *VRaptor* e o *Hibernate* se encarregarão de fazer o tratamento das informações.

```
package br.com.projetoPosVRaptor.model;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
@Entity
public class Endereco {
	@Id
	@GeneratedValue
	private int id;
	private String logradouro;
	private int num;
	private String complemento;
	private String cidade;
	private String estado;
	private String cep;
	// getter e setters e métodos de negócio que julgar necessário
}
```

Figura 36. Classe Endereço com anotações – *VRaptor*

7.5 A classe *Controller*, o coração da aplicação no *VRaptor*

O *VRaptor*, procura a simplicidade e facilidade no desenvolvimento das ferramentas. Ao contrário do *Servlets*, o *VRaptor* procura englobar todas funcionalidades do objeto em um local (classe), sendo a *Controller* a camada de trabalho do *VRaptor*, onde funciona todas as requisições à aplicação. É nela que se insere toda a lógica de negócio do sistema.

Por convenção do *framework*, sempre se utiliza o nome da classe sucedido da expressão '*controller*'. Por exemplo, se está trabalhando com uma classe modelo 'Cliente' o controlador chamar-se-á '*ClienteController*' sempre com a anotação da classe com

'@Resource', que faz a indicação ao *VRaptor* que essa classe irá expor *URIs* para acesso via navegador web, expondo recursos da aplicação.

Veja abaixo o exemplo de criação dessa classe:

```
package br.com.projetoPosVRaptor.controller;

import java.util.List;

import br.com.caelum.vraptor.Path;
import br.com.caelum.vraptor.Resource;

@Resource
@Path("/cliente")
public class ClienteController{

}
```

Figura 37. Classe ClienteController - *VRaptor*

OBS: A classe '*ClienteDao*' é a responsável pela obtenção das informações. No exemplo acima, se entende que a classe '*DAO'* para o cliente já existe um método '`pegaTodos()`' que faz essa conexão à um banco de dados e captura as informações solicitadas.

Observa-se abaixo um exemplo dessa classe *ClienteDAO.java*, aproveitando o gerenciamento de dependências do *VRaptor*, usa-se a anotação '*@Component*'. Assim o *VRaptor* se encarregará de criar o *DAO* e utilizá-lo na classe:

```
package br.com.projetoPosVRaptor.dao;

import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
import br.com.caelum.vraptor.ioc.Component;
import br.com.projetoPosVRaptor.infra.CriadorDeSession;
import br.com.projetoPosVRaptor.model.Cliente;
```

```
@Component
public class ClienteDao {

	private Session session;

	public ClienteDao(Session session){
		this.session = CriadorDeSession.getSession();
	}

    public List<Cliente> pegaTodos() {
	try{
		return this.session.createCriteria(Cliente.class).list();
	}finally{
		CriadorDeSession.fecha();
	}
    }
}
```

Figura 38. Classe ClienteDao - *VRaptor*

Note no código anterior, que estamos utilizando a classe '*CriadorDeSession*' para gerar uma sessão de conexão ao banco utilizando a classe '*ClienteDao*', classe essa que obtem/insere as informações no banco de dados:

```
...
	public ClienteDao(Session session){
		this.session = CriadorDeSession.getSession();
	}
...
```

Figura 39. Classe ClienteDao (detalhe sessão) – *VRaptor*

Como estamos trabalhando com o *Hibernate*, precisamos solicitar que o mesmo encapsule uma conexão para que possamos utilizar. Esse objeto, que precisamos é uma '*Session*', do pacote *org.hibernate*. A seguir, criamos a classe '*CriadorDeSession.java*' e instanciamos a classe do *Hibernate* de nome '`org.hibernate.cfg.AnnotationConfiguration`':

```
package br.com.projetoPosVRaptor.infra;

import javax.annotation.PreDestroy;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;
import br.com.caelum.vraptor.ioc.Component;
import br.com.caelum.vraptor.ioc.ComponentFactory;

@Component
public class CriadorDeSession implements ComponentFactory<Session> {
private static Session session;

	@Override
	public Session getInstance() {
		return null;
	}

	public static Session getSession() {
		AnnotationConfiguration configuration = new
					AnnotationConfiguration();
		configuration.configure();
		SessionFactory factory = configuration.buildSessionFactory();
		session = factory.openSession();
		return session;
	}

	//Tratamento de Sessão - fecha
	@PreDestroy
	public static void fecha(){
		session.close();
	}
}
```

Figura 40. Classe CriadorDeSession - *VRaptor*

O objeto criado nesta classe acima é muito importante, pois encapsula todas chamadas ao banco de dados, que facilita a maneira de tratamento desse acesso.

A configuração da conexão ao banco de dados, assim como mapeamento das classes '*model*' contidas no projeto para o *Hibernate*, serão definidas no arquivo *XML*: '`hibernate.cfg.xml`' que deve ficar localizado na pasta 'src' do projeto. No caso do projeto

em questão ficaria no caminho ‘ProjetoPosVRaptor > src’ e está configurado para o banco de dados *MySQL*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
       <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/projetopos</property>
       <property
name="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</property>
     <!-- USUARIO BD -->
     <property name="hibernate.connection.username">root</property>
     <!-- SENHA BANCO -->
    <property name="hibernate.connection.password"></property>

    <!-- GERA TABELAS SE ELAS NÃO EXISTIREM! -->
    <property name="hibernate.hbm2ddl.auto">update</property>

    <!-- Exibe instruções SQL gerado e formatado no Console  -->
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.format_sql">true</property>

    <!-- Mapeamento das classes TO persistentes  -->
    <mapping class="br.com.projetoPosVRaptor.model.Cliente" />
    <mapping class="br.com.projetoPosVRaptor.model.Endereco" />
 </session-factory>
</hibernate-configuration>
```

Figura 40. hibernate.cfg.xml - *VRaptor*

Como foi explicado anteriormente, é necessário mapeamento das classes do projeto pelo *Hibernate*. Devemos configurar no arquivo ‘`hibernate.cfg.xml`’ todas as classes que fazem parte do modelo projeto, para que o framework entenda que tais classes serão utilizadas no gerenciamento dos dados. Assim, informamos a classe ‘*Cliente*’ e ‘*Endereco*’ e o caminho completo do pacote em que as classes estão alocadas no projeto, conforme abaixo:

```
...
      <!-- Mapeamento das classes TO persistentes  -->
      <mapping class="br.com.projetoPosVRaptor.model.Cliente" />
      <mapping class="br.com.projetoPosVRaptor.model.Endereco" />
...
```

Figura 41. hibernate.cfg.xml (detalhe) - *VRaptor*

Assim como no projeto '*Jsp/Servlets*', é necessária a inclusão do arquivo de conexão ao banco de dados na pasta '*WebContent* > *WEB-INF* > *lib*' do projeto. Podemos inclusive utilizar o mesmo arquivo de conexão (*mysql-connector-java-5.1.18-bin.jar*), utilizado no projeto '*Jsp/Servlets*', para essa funcionalidade.

Ao final, vamos instanciar a classe *ClientesController.java* e o método '`lista()`'que nos retornará a lista de clientes:

```
package br.com.projetoPosVRaptor.controller;
import java.util.List;
import br.com.caelum.vraptor.Path;
import br.com.caelum.vraptor.Resource;
import br.com.projetoPosVRaptor.dao.ClienteDao;
import br.com.projetoPosVRaptor.model.Cliente;

@Resource
@Path("/cliente")
public class ClienteController{
       private final ClienteDao dao;
       public ClienteController(ClienteDao dao) {
         this.dao = dao;
    }
       @Path("/lista")
       public List<Cliente> lista(){
              return dao.pegaTodos();
       }
}
```

Figura 42. ClienteController com método 'lista( )' – *VRaptor*

Uma observação importante, é que nota-se que não há nenhuma '*query*' para a consulta ao banco, tudo é feito automaticamente pelo *Hibernate*. Mais uma facilidade que o *VRaptor* utiliza.

7.6 *URI* e *View* visualização das informações

A anotação *'@Resource'* na classe controladora do método tem o papel fundamental. Como visto anteriormente, cria-se a classe *'ClienteController'* e nela o método 'lista'. Ao digitar no navegador de internet 'nomedoprojeto/produtos/lista' o *VRaptor* direciona automaticamente as requisições à *URI*.

Assim, se pode notar que a convenção para a criação das *URIs* será sempre: '/<nome_controller>/<nome_metodo>'.

Todavia, ao terminar a execução do método, o *VRaptor* fará o `dispatch` da requisição para o jsp `/WEB-INF/jsp/cliente/lista.jsp`. Ou seja, a convenção para a view padrão é `/WEB-INF/jsp/<nome_do_controller>/<nome_do_metodo>.jsp`.

O método '*lista*' nos traz a lista de clientes, mas para o seu acesso na *Jsp*, o *VRaptor* exporta um método para a jsp por atributos dessa requisição. Assim, no *ClientesController* o método '*lista*' usará o atributo '*clienteList'* que contem a lista como retorno:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
      pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Lista Clientes</title>
</head>
<body>
      <h3>Relação de Clientes</h3>
      <table border=1>
```

```
            <tr>
                  <td width="300px">Nome</td>
                  <td width="150px">Cidade</td>
                  <td width="20px">Estado</td>
            </tr>
            <c:forEach items="${clienteList}" var="cliente">
                  <tr>
                        <td>${cliente.nome}</td>
                        <td>${cliente.endereco.cidade}</td>
                        <td>${cliente.endereco.estado}</td>
                  </tr>
            </c:forEach>
      </table>
</body>
</html>
```

Figura 43. lista.jsp – *VRaptor*

Obs: É importante o uso da *taglib* para funcionamento da aplicação. No topo da *jsp*, importá-la como abaixo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

Figura 44. lista.jsp - *VRaptor*

A convenção para esse tratamento dos atributos é simples, no caso de uma *collection*, como anteriormente é sempre precedido de '*List*'. Ex: '*nomeCollectionList*'. Se o método retorna a classe *model* completa, então o atributo que será exportado será o nome da classe. Ex: Cliente, o atributo utilizado será cliente.

> As *taglibs* são *tags* customizadas *Java*, que ajudam a eliminar consideravelmente a redundância de códigos nas páginas *jsp*, através da biblioteca *JSTL* (*JavaServer Pages Standart Tag Library*), de acordo informações do site da *Oracle* (atual desenvolvedora da tecnologia *Java*) (Oracle. JSTL, 2012).

Ao executar a aplicação, a mesma será exibida no *browser* e finalmente é possível visualizar em tela todos os Clientes e seu Endereço (cidade e estado) cadastrados no Banco de Dados da aplicação, como imagem a seguir:

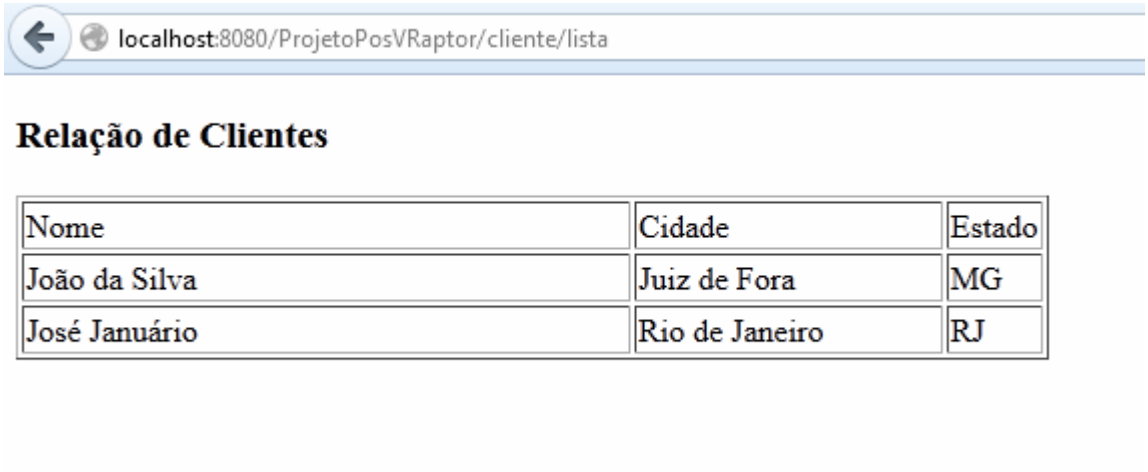

Figura 45. Visualização do projeto *VRaptor* em execução

Como pode ser observado, o resultado final da aplicação será idêntico ao resultado obtido com *Jsp/Servlets*.

8 Comparativo entre *Servlets/Jsp* e *VRaptor*

Consolidada no Mercado e uma das pioneiras no desenvolvimento *web* com *Java*, a utilização de *Jsp/Servlets* vem caindo em desuso, visto à grande quantidade de procedimentos 'braçais' utilizados no desenvolvimento de aplicações.

Como alternativa, são utilizados vários recursos para prolongar a vida do *Jsp*/*Servlets*, como *javascripts* e inclusão de bibliotecas específicas.

Porém alguns *frameworks* vêm se tornando atraentes, e de fácil desenvolvimento, levando a uma melhor análise das novidades e ferramentas *Java*.

O *VRaptor* consegue unir vários *frameworks* que já são referência no Mercado em um só, como *Spring Security, Hibernate* entre outros. Não que tais ferramentas não sejam possíveis de se utilizar com a linguagem *Jsp/Servlets*, porém, em um único pacote, sem necessidade de configurações específicas, facilita muito a vida do desenvolvedor.

Observou-se que os códigos utilizados com o *VRaptor* reduzem, e muito, os códigos empregados, os princípios utilizados com a ajuda do *Hibernate* (que integra o *VRaptor*) retira completamente a necessidade de digitar até mesmo desde pequenas, como grandes *querys SQL*, no fonte do projeto, otimizando o processo.

Ao contrário do desenvolvimento de uma página *web* utilizando o *Jsp/Servlets* básico, uma grande vantagem em se usar o *VRaptor* é a organização e disponibilidade dos recursos que a ferramenta proporciona, separando e organizando melhor cada parte da aplicação.

Adotando o mesmo servidor de aplicação '*Apache TomCat'*, pode-se reescrever as aplicações *Jsp/Servlets* manuseando o *VRaptor* sem afetar a estrutura já pronta para uso.

Com o eminente desuso do *Jsp/Servlet*, obriga-se a procura de alternativas eficazes e de baixo custo, tanto no desenvolvimento como no aprendizado. Como escolha e uma comunidade ativa, o *VRaptor* tem se destacado no mercado, e sendo uma boa opção para a categoria *Java*.

8.1 Quadro Comparativo

Tecnologia Ref.	*JSP / Servlets*	*VRaptor*
Tamanho de Código	Maior quantidade de código fonte, visto que o funcionamento *JSP* > *servlet* > *JSP* reaproveita poucas telas. O problema pode ser amenizado com o uso de JavaScripts e JQuery.	Com o uso do *MVC*, o *VRaptor* simplifica o fonte, reduzindo duplicidade de código e separando melhor a parte de visão da parte lógica se aproximando ao entendimento do *JSF*.
Tempo de Criação	Em vista da maior quantidade de fonte, decorre um maior tempo de desenvolvimento, testes e atenção.	Com um número menor de linhas de código fonte e funcionalidades mais simples, simplificam o desenvolvimento e testes.
Tempo de Aprendizado	Quando desenvolvido com experiência em *Java*, facilita o entendimento, assim como conhecimento de *html*.	Quando desenvolvido com experiência em *Java*, facilita o entendimento, assim como conhecimento de *html*.
Organização	A funcionalidade dos *Servlets*: é criado um para cada ação, o que pode aumentar a quantidade de classes e até mesmo confundir o desenvolvedor.	A funcionalidade do *Controller*: é simplificar o desenvolvimento, podendo englobá-los em uma única classe, organizando-os melhor.

Design	Os *Servlets/JSP* não possuem uma biblioteca nativa para *view* (visão), a não ser pela inclusão de componentes do próprio *HTML/CSS* e/ou *JQuery*.	O *VRaptor* possui bibliotecas nativas para utilização em suas telas e possibilita ampliações à outros frameworks de terceiros.
Segurança	Não possui nenhum *Framework* e o processo deve ser feito manualmente pelo desenvolvedor.	Acompanha bibliotecas para segurança da aplicação, como o *Spring*, dando agilidade e confiabilidade.
Banco de Dados	Por utilizar uma conexão direta, pode-se gerenciar a chamada ao banco e desconexão.	Quando se utiliza o *Hibernate*, o mesmo gerencia as conexões no banco, que apesar do controle na aplicação, deixa processos ativos no mesmo.
Atualizações	Não existem. Pode ser 'atualizado' com *frameworks* de terceiros dando uma sobrevida.	Comunidade ativa e brasileira, sempre aberta a sugestões e críticas. Atualizações do framework são frequentes.

9 Considerações Finais

A tecnologia *Jsp/Servlets* básica para desenvolvimento *web* está cada vez mais caindo em desuso. Apesar da sobrevida com alguns *frameworks* implementando novas funcionalidades ao *Jsp/Servlets*, como, por exemplo, a *JQuery* e até mesmo o *Hibernate*, vê-se um mercado sempre em busca de novidades com a facilidade no desenvolvimento de aplicações e do lucro imediato. Com o *VRaptor* surge uma opção à frente dessa tecnologia para dinamizar o desenvolvimento *web*, sem complicadas instalações e/ou incompatibilidade de ferramentas.

Porém, o *VRaptor* foi e ainda é desenvolvido pela *Caelum*, escola responsável inclusive, pela sua divulgação. Contudo uma dúvida nos preocupa, a ferramenta tende a se tornar um padrão de desenvolvimento com o peso de um "*JSF*", por exemplo, ou cair no desuso e abandono?

No momento o *VRaptor* é uma tendência de mercado e seu fórum encontra-se bem movimentado e ativo, onde com mais de 150.000 usuários, o *GUJ* (www.guj.com.br) possui uma sessão específica para frameworks e bibliotecas brasileiras em seu fórum e empresas como Locaweb (www.locaweb.com.br), Grupo Defferrari (www.defferrari.com.br) e o Instituto Metodista Granbery (www.granbery.edu.br) possuem projetos ativos com esta ferramenta.

Todavia, assim como o *VRaptor* é uma tendência de mercado, outros frameworks como o próprio *Jsp/Servlets* e o *JSF* na versão 1.2, já foram tendência e ainda assim estão perdendo espaço por versões mais aprimoradas.

Infelizmente o caminho das tecnologias de programação busca se orientar, geralmente, ao que é referência. Cabe à ferramenta manter-se ativa, sempre com novas versões e *upgrades*.

Enfim, a atualização profissional é o equilíbrio de futuras escolhas, principalmente ao iniciarmos um projeto, para que o mesmo não seja prejudicado por uma tecnologia muito atual, sem perspectivas e ou sem fontes de pesquisa. Por outro lado, utilizar uma ferramenta defasada também compromete o trabalho.

A velocidade de novidades no meio de desenvolvimento de sistemas é constante, fazendo com que dificulte a inclusão de pessoal capacitado e comprometido tanto em relação às novas ferramentas quanto nas mais antigas.

REFERÊNCIAS

Goncalves, Edson. **Desenvolvimento de aplicações web com jsp, servlets, javaserver pages, hibernate, ejb3 persistence e ajax** (2007).

LUCKOW, Décio Heinzelmann*;* MELO, Alexandre Altair de. **Programação Java para a Web**: aprenda a desenvolver uma aplicação financeira pessoal com as ferramentas mais modernas da plataforma Java. São Paulo: Novatec, 2010.

H. M. Deitel; P. J. Deitel. **Java**: como programar.8.ed. Pearson, 2010.

KURNIAWAN, BUDI. **Java para a Web com Servlets, JSP e EJB**. Ciência Moderna, 2002, Ed.1.

MOTTA, Luciana Campos. **Framework** - definição. Disponível em: <http://www.frameworkdemoiselle.gov.br/framework/sobre/>. Acesso em 23/02/2012.

Caelum. **Desenvolvimento ágil para web 2.0 com vraptor, hibernate e AJAX**. Disponível em: <http://www.caelum.com.br/download/caelum-java-web-vraptor-hibernate-ajax-fj28.zip>. Acesso em: 23/02/2012.

Caelum. **VRaptor**. Disponível em: <http://vraptor.caelum.com.br/pt/ >. Acesso em: 01/09/2013.

Caelum. **VRaptor3 -** o guia inicial de 10 minutos. Disponível em: <http://vraptor.caelum.com.br/pt/docs/guia-de-dez-minutos>. Acesso em: 09/01/2013.

Caelum. VRaptor3 – o guia inicial de 1 minuto. Disponível em: <http://vraptor.caelum.com.br/pt/docs/guia-de-um-minuto>. Acesso em: 09/01/2013.

JavaFree.org. **Java.** Disponível em: <http://javafree.uol.com.br/wiki/Java>. Acesso em: 10/02/2012.

MADEIRA, Marcelo. **Vantagens do vRaptor.** Disponível em: <http://celodemelo.wordpress.com/2007/05/12/vantagens-do-vraptor/>. Acesso em: 10/02/2012.

COELHO, Igor. **Desenvolvimento web em java com JSP e Servlets.** Disponível em: <http://www.slideshare.net/igocoelho/jspservlets-fatene201006-4624684> Acesso em: 02/03/2012.

ALMEIDA, Adriano. **Java EE6**: começando com as Servlets 3.0. Disponível em: <http://blog.caelum.com.br/java-ee6-comecando-com-as-servlets-3-0/> Acesso em: 02/03/2012.

Caelum. **Java Web FJ-21**. Disponível em: <http://www.caelum.com.br/download/caelum-java-web-fj21.pdf>.Acesso em: 04/03/2012.

SILVA, Leandro. **Introdução à taglib**. Disponível em: <http://www.devmedia.com.br/introducao-a-taglib/3317>. Acesso em: 30/10/2012

Oracle Tecnology network. **JavaServer pages standart.** Disponível em: <http://www.oracle.com/technetwork/java/index-jsp-135995.html>. Acesso em: 30/10/2012.
Oracle. **O que é a tecnologia Java e por que é necessária**. Disponível em: <http://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em: 01/09/2013.

Softwarelivre.org. **Quer alta produtividade no desenvolvimento web em java? Experimente o VRaptor3**. Disponível em: <http://softwarelivre.org/revistaespiritolivre/blog/quer-alta-produtividade-no-desenvolvimento-web-em-java-experimente-o-vraptor3>. Acesso em: 01/12/2012.

AGOSTI, Cristiano. **Desenvolvimento Web em Java: vRaptor**. Disponível em: http://www.slideshare.net/cristianoagosti/vraptor-3. Acesso em: 05/10/2013.