02

Configurando primeiro projeto com VRaptor 4

Olá! Bem vindo ao curso de VRaptor 4!

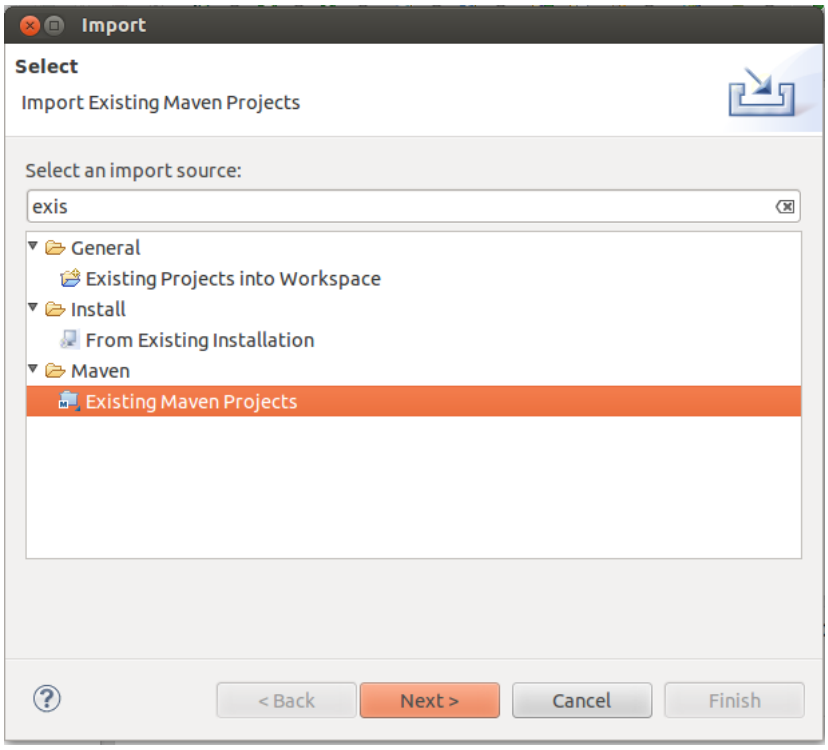
Sobre o VRaptor 4

O VRaptor 4 é a mais nova versão desse framework que te traz muita produtividade no desenvolvimento java web! Pensando em melhorar ainda mais essa área do desenvolvimento java, a Caelum criou e mantém o VRaptor. Nessa nova versão temos os poderosos recursos do CDI (http://www.cdi-spec.org) integrados ao nosso core, afinal essa especificação está cada vez mais ganhando espaço no mercado. Nesse curso, aprenderemos os principais recursos iniciais do VRaptor, como controle de fluxo de uma requisição web, validação de dados, interceptors, injeção de dependências, integração com plugins, entre outros recursos e boas práticas de desenvolvimento.

Download e configuração do projeto

Como ponto de partida, precisamos baixar a base do projeto que usaremos durante o curso. Esse projeto trabalhará no contexto de controle de estoque de produtos. Esse é um cenário simples, porém que nos permitirá usar muitos dos poderosos recursos dessa nova versão do framework! É só você clicar aqui (https://github.com/alura-cursos/desenvolvimento-web-com-vraptor-4/archive/master.zip) para fazer o download do projeto.

Pronto! Agora que você já fez o download, basta extrair do zip e importar o projeto no eclipse. Pela facilidade de uso, vamos usar o maven para gerenciamento de nossas dependências. Se você não conhece o maven, você pode gostar do nosso curso voltado para essa ferramenta (http://www.alura.com.br/cursos-online-java/maven). No Eclipse, basta você selecionar a opção `File > Import` e selecionar a opção `Import (Existing Maven Project into workspace)`.



Repare que é um projeto Java Web normal. Temos por enquanto as classes `Produto`, que só possui um `id`, `nome`, `valor` e `quantidade`:

```
package br.com.caelum.vraptor.model;

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

@Entity
public class Produto {

    @GeneratedValue @Id
    private Long id;

    private String nome;
    private Double valor;
    private Integer quantidade;

    // getters e setters
}
```

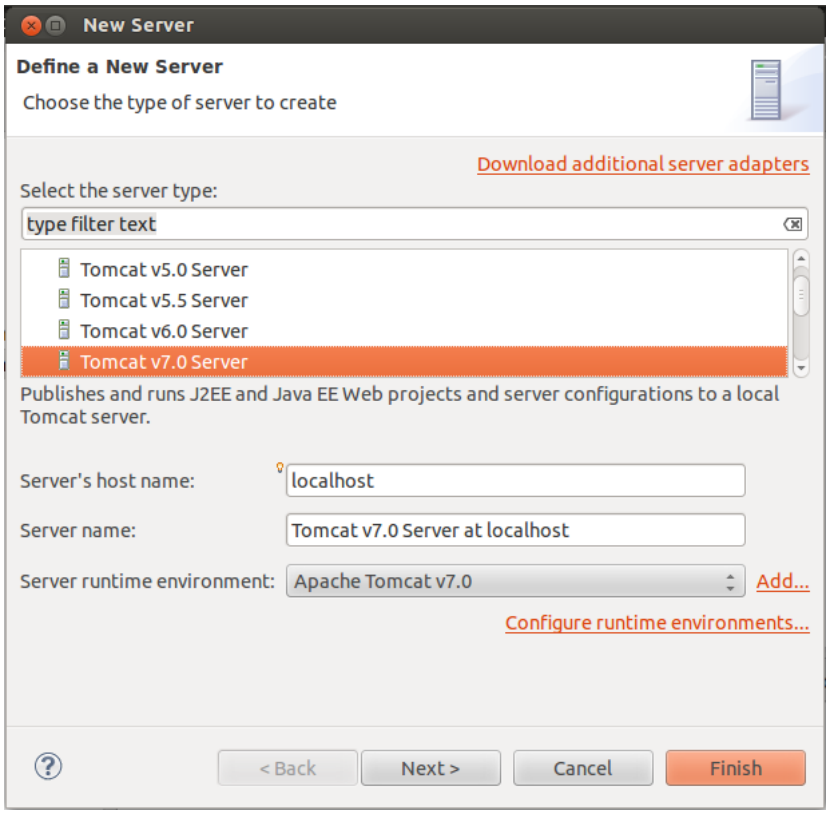
E também temos a classe `ProdutoDao`, que vai isolar nosso acesso ao banco de dados. Para nos aproximar de uma aplicação real do dia a dia, vamos usar o JPA com hibernate (http://www.alura.com.br/cursos-online-java/jpa) que é muito usado pelo mercado para auxiliar na persistência dos dados. Não se preocupe, você não precisa ter nenhum conhecimento dessa ferramenta para prosseguir o curso. Repare que nossa classe `Produto` possui algumas anotações para indicar que é uma tabela do banco de dados e que seu id é auto gerado, e usaremos a classe `EntityManager` do JPA para fazer nosso CRUD (cadastro, leitura, alteração e remoção de produtos), ele nos dá uma forma mais orientada a objetos de fazer essas operações, assim trabalhamos em um ambiente mais familiar do que fazer as queries na mão e programar pensando de forma relacional.

```
public class ProdutoDao {

    private final EntityManager em;

    public ProdutoDao(EntityManager em) {
        this.em = em;
    }

    public void adiciona(Produto produto) {
        em.persist(produto);
    }

    public void remove(Produto produto) {
        em.remove(produto);
    }

    public void busca(Produto produto) {
        em.find(Produto.class, produto.getId());
    }

    @SuppressWarnings("unchecked")
    public List<Produto> lista() {
        return em.createQuery("select p from Produto p")
            .getResultList();
    }
}
```

Instalando o Tomcat 7

Vamos usar o tomcat como container web para executar nossa app de controle de estoque. Se você estiver mais adaptado com outro container ou servidor de aplicação, você pode preferir usá-lo, porém é importante lembrar que o VRaptor 4 usa recursos do JavaEE 7, como o CDI 1.1, portanto seu servidor precisará suportar essa versão da especificação. Além do Tomcat e o Jetty (a partir da versão 8), alguns dos servidores já testados e suportados são o Glassfish 4 e WildFly 8.0.

Você pode baixar o Apache Tomcat 7 por esse link (http://tomcat.apache.org/download-70.cgi). Depois de baixar, apenas extraia o zip para uma pasta que preferir em seu computador, e em seguida clique com o botão direito dentro da aba (view) Server. Deve aparecer uma caixa de dialogo com a opção New Server, vamos selecionar essa opção e em seguida escolhemos a opção Apache Tomcat 7. Agora precisamos informar o local em que a pasta do tomcat foi descompactada:



Pronto! Agora que finalizamos, já temos o `Tomcat` como opção de servidor da aba `Server`.

Primeiro Controller do VRaptor 4

A principio queremos que ao acessar a uri `"/"` sejamos direcionados para uma página html simples com uma mensagem dizendo que esse mapeamento deu certo. Para isso, vamos criar a classe `ProdutoController` no pacote `br.com.caelum.vraptor.controller` e adicionar a anotação `@Controller`. Ao adicionar essa anotação estamos ensinando ao VRaptor que a função dessa classe será receber as informações que chegam pela requisição http do usuário, ela quem vai determinar que ao acessar uma url alguma regra de negócio de nossa aplicação deverá ser executada.

```
@Controller
public class ProdutoController {

}
```

Vamos adicionar um método chamado `inicio`, que receberá a anotação `@Path("/")` indicando seu caminho.

```
@Controller
public class ProdutoController {

    @Path("/")
    public void inicio() {

    }
}
```

Está quase tudo pronto! Agora só precisamos criar o arquivo `inicio.jsp` que possui um html simples de introdução ao projeto:

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Projeto de Produtos</title>
    </head>
    <body>
        <h1>Primeira lógica com VRaptor 4!</h1>
    </body>
</html>
```

Mas como ensinar ao VRaptor que ao acessar a uri `"/"` ele deve encaminhar para o arquivo `inicio.jsp` ? A resposta é muito simples, basta adicionar esse arquivo na pasta `WEB-INF/jsp/produto/` . Por convenção o VRaptor espera que ao acessarmos o método `inicio` da classe `ProdutoController` , queremos utilizar a página chamada `inicio` (com o mesmo nome do método) que está dentro da pasta `produto` (que é o nome da classe, sem o sufixo controller).

Em outras palavras, o nome do método reflete no nome do arquivo jsp, e o nome do controller (sem o sufixo controller) reflete no nome da pasta em que esse arquivo está. Portanto neste caso, basta adicionar o arquivo no caminho `/WEB-INF/jsp/produto/inicio.jsp` .

Vamos testar para garantir que tudo está funcionando. Para executar esse nosso primeiro controller, adicionamos o projeto no tomcat clicando com o botão direito do mouse e selecionando a opção `Add and Remove` . Selecione o projeto `vraptor-produtos` e clique em `Finish` . Click no icone de start ou utilize o atalho `CTRL+ALT+R` para subir o servidor.

Pronto! Basta acessar o link http://localhost:8080/vraptor-produtos/ (http://localhost:8080/vraptor-produtos/) para ver nossa primeira lógica!

Criando uma listagem de produtos

Agora que já conhecemos algumas convenções do VRaptor, vamos exercitar nosso conhecimento criando uma listagem simples de produtos. Para isso, vamos adicionar um método chamado `lista` na classe `ProdutoController` .

```
@Controller
public class ProdutoController {

    @Path("/produto/lista")
    public List<Produto> lista() {
        EntityManager em = JPAUtil.criaEntityManager();
        ProdutoDao dao = new ProdutoDao(em);
        return dao.lista();
```

```
    }
}
```

Para simplificar o código, isolamos a regra de criação da classe `EntityManager` do JPA na classe `JPAUtil`:

```
public class JPAUtil {

    public static EntityManager criaEntityManager() {
        EntityManagerFactory factory = Persistence
                .createEntityManagerFactory("default");
        return factory.createEntityManager();
    }
}
```

Complicado? Não se preocupe, o VRaptor ainda vai nos ajudar e muito em todo esse trabalho!

Repare que nosso método `lista` está retornando uma lista de produtos, mas como acessar essa lista em nossa JSP? Podemos fazer uso da Expression Language (EL) para recuperar o valor na JSP, algo como: `${nomeDaVariavelAqui}`. Mas qual o nome da variável?

Por convenção, o nome da variável disponibilizada em nossa JSP será `produtoList`, ou seja, sempre o nome da classe com a primeira letra minúscula e o sufixo `List`. Já se o método retornasse um único Produto, o nome da variável disponibilizada na JSP seria apenas `produto`.

Sabendo disso, podemos agora criar a `lista.jsp` dentro da pasta `WebContent/WEB-INF/jsp/produto`.

Nesse arquivo vamos apenas criar uma tabela com o nome, valor e quantidade de produtos. Para preencher o corpo dessa tabela, vamos iterar por essa lista de produtos, em EL `${produtosList}`.

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

<html>
    <head>
        <title>Lista de Produtos</title>
    </head>
    <body>
        <table>
            <thead>
                <tr>
                    <th>Nome</th>
                    <th>Valor</th>
                    <th>Quantidade</th>
                </tr>
              </thead>
              <tbody>
              <c:forEach items="${produtoList}" var="produto">
                <tr>
                    <td>${produto.nome}</td>
                    <td>${produto.valor}</td>
                    <td>${produto.quantidade}</td>
                </tr>
            </c:forEach>
            </tbody>
```

```
        </table>
    </body>
</html>
```

Para estilizar essa página sem nos preocuparmos muito em ficar criando css, vamos usar o twitter bootstrap (http://getbootstrap.com/). Basta adicionar seu import no inicio do seu html, e adicionar as classes dele de formatação da tabela. Para centralizar, também vamos criar uma `div` com classe container que o bootstrap fará todo o trabalho.

```
<html>
    <link rel="stylesheet" type="text/css" href="../bootstrap/css/bootstrap.css">
    <body>
        <div class="container">

        <h1>Listagem de Produtos</h1>

        <table class="table table-stripped table-hover table-bordered">
            <!-- restante do código omitido -->
        </table>
    </body>
</html>
```

Comó já temos alguns produtos pré cadastrados, vamos reiniciar o tomcat e acessar o link http://localhost:8080/vraptor-produtos/produto/lista (http://localhost:8080/vraptor-produtos/produto/lista) para ver o resultado!