Workshop 01 – Livraria Web

O objetivo deste workshop é consolidar o conhecimento adquirido durante o curso. Será desenvolvido uma Livraria virtual, onde , nesse primeiro workshop, é possível consultar livros, listá-los e ver seus detalhes como autor, preço e descrição.

Exercícios

Exercício 1: Criando a classe Livro

Exercício 2: Criando o Banco de dados

Exercício 3: Criando a classe LivroDao

Exercício 4: Criando o Bean de pesquisa

Exercício 5: Desenvolvendo a página inicial

Exercício 6: Desenvolvendo a página de resultados

Exercício 7: Desenvolvendo a página Livro

Exercício 8: Cadastrano novos Livros ná página

Exercício 1 – Criando a classe Livro

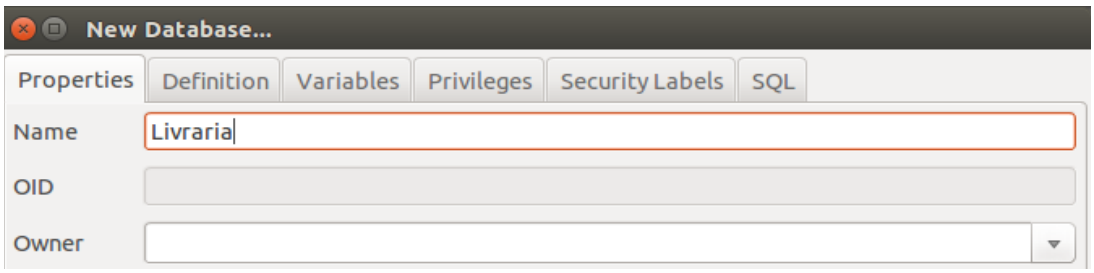
1. Vamos utilizar o projeto **Livraria-web** criado no laboratório de Maven. Caso o projeto tenha sido usado, recomenda-se criar um novo, do zero, para não bagunçar a estrutura do seu projeto. Lembre -se de configurar o **pom.xml** do seu projeto, acrescentando os repositórios do postgresql e javax.faces para rodar os exercícios a seguir.

```
<!-- Postgresql Driver -->

	<dependency>
		<groupId>postgresql</groupId>
		<artifactId>postgresql</artifactId>
		<version>9.1-901-1.jdbc4</version>
	</dependency>
<!-- Mojarra (implementação do JSF 2) -->
	<dependency>
		<groupId>org.glassfish</groupId>
		<artifactId>javax.faces</artifactId>
		<version>2.2.12</version>
		<scope>compile</scope>
	</dependency>
```

2. A classe **Livro** representa a entidade livro em nosso projeto. Obedecerá o padrão de projeto VO (Value Object) e conterá os dados de um livro. Crie a classe **Livro** (dentro do pacote **model**) assim como apresentado no UML abaixo. Não se esqueça de acrescentar os **getters** e **setters**.

Exercício 2 – Criando o Banco de dados

1. Para o projeto funcionar, precisaremos de um banco de dados para guardar informações, como os livros que temos em estoque, os clientes cadastrados no site e os pedidos de cada cliente. Primeiro, crie um novo Banco de dados no **Postgres.** Conecte ao servidor usado, clique com o botão direito em **Databases** e em seguida, clique em **New Database...** Vamos chamar nosso novo banco de dados de Livraria, para deixar explicito o banco que estamos usando. Feito isso, clique **Ok.**



2. Agora, precisamos criar a conexão entre o banco de dados e o seu projeto. Crie a classe **FabricaConexao** assim como passado no laboratório de Banco de dados, e nessa classe, altere somente a url de destino da conexão.

```
static final String url = "jdbc:postgresql://localhost:5432/Livraria";
```

3. Crie também a classe **TestaConexao** para fazer o teste, ver se está tudo em ordem.

4. Dentro do seu Banco de dados Livraria, vamos agora criar tabelas onde vão ser inseridas e recolhidas as informações necessárias.

   Primeiro, criaremos nosso **ESTOQUE** de livros.

```
CREATE TABLE estoque (
        COD_LIVRO SERIAL NOT NULL,
        TITULO VARCHAR(30) NOT NULL,
        AUTOR VARCHAR(20) NOT NULL,
        PRECO NUMERIC NOT NULL,
        IMAGEM VARCHAR(80) NOT NULL,
        DESCRICAO VARCHAR(80),
        PRIMARY KEY (COD_LIVRO));

INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('GRANDE SERTAO -
VEREDAS', 'ROSA, JOAO GUIMARAES', 165, 'imagens/veredas.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('QUANDO NIETZSCHE
CHOROU', 'YALOM, IRVIN D.', 49.9, 'imagens/chorou.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('CASSINO ROYALE - JAMES
BOND 00', 'Fleming, Ian', 29.9, 'imagens/james.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('FILOSOFIA DO TEDIO',
'Svendsen, Lars', 29.9, 'imagens/tedio.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O CASAMENTO',
'Rodrigues, Nelson', 39.9, 'imagens/casamento.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('NEVE', 'PAMUK, ORHAN',
54, 'imagens/neve.jpg');
```

```
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('VOLTA AO MUNDO EM
OITENTA DIAS', 'VERNE, JULIO', 16.5, 'imagens/volta_mundo.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('CRISTOVAO COLOMBO',
'VERNE, JULIO', 16.5, 'imagens/cristovao_colombo.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('VINTE MIL LEGUAS
SUBMARINAS', 'VERNE, JULIO', 14.9, 'imagens/submarinas.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O SENHOR DOS ANEIS',
'TOLKIEN, J.R.R.', 169.9, 'imagens/senhor.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('HARRY POTTER',
'ROWLING, J.K.', 89.7, 'imagens/harry.png');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('A AVENTURAS DE PI',
'MARTEL, YANN', 23.5, 'imagens/lifeofpi.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('PARA ONDE ELA FOI?',
'FORMAN, GAYLE', 20.0, 'imagens/onde.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O LIVRO DO CEMITERIO',
'GAILMAN, NEIL', 20.0, 'imagens/cemiterio.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES (SANDMAN VOL 1',
'GAILMAN, NEIL', 489.0, 'imagens/sandman.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('WATCHMEN', 'MOORE,
ALAN', 37.4, 'imagens/watchmen.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('JUSTICEIRO NOIR', 'TIIER,
FRANK', 12.5, 'imagens/justiceiro.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('SUPERMAN', 'TOMASI,
PETER', 5.9, 'imagens/superman.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('BATMAN', 'SNYDER,
SCOTT', 5.9, 'imagens/batman.jpeg');
```

Nesse código sql, estamos criando uma tabela **estoque** e nela estamos adicionando os livros, seus titulos, autores, preço, caminho de imagem e futuramente, uma descrição para cada livro.

Exercício 3 – Criando a classe LivroDao

1. Primeiro iremos definir a interface GenericDao afim de mantermos um padrão para todas as classes DAO que sejam construídas em nosso projeto.

```
package dao;


import java.io.Serializable;
import java.util.Collection;

public interface Dao<E, K extends Serializable> {

    /**
     * Busca a Serializable pelo seu identificador.
     *
     * @param identificador da Serializable
     *
     * @return Serializable pesquisada
     */
    E consultar(final K id);
```

```
    /**
     * Altera a Serializable.
     *
     * @param Serializable
     */
    void alterar(final E Serializable);

    /**
     * Insere a Serializable.
     *
     * @param Serializable
     */
    void salvar(final E Serializable);

    /**
     * Remove a Serializable.
     *
     * @param Serializable
     */
    void remover(final E Serializable);

    /**
     * Lista todos os objetos da Serializable.
     *
     * @return Collection<E>
     */
    Collection<E> listar();
     }
```

2. Agora iremos criar a classe **LivroDao**. Essa classe será responsável pelo acesso ao banco de dados e retornar dados referentes a classe **Livro**. Crie a classe LivroDao (dentro do pacote **dao**) para acessar o banco de dado e fazer as operações de consulta, um livro e uma lista de livros. Como exemplo segue uma parte do código abaixo.

```
package dao;

import java.sql.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.logging.Logger;

import service.FabricaConexao;
import service.Livro;

public class LivroDao implements Dao<Livro,Integer> {
    Logger LOG = Logger.getGlobal();
```

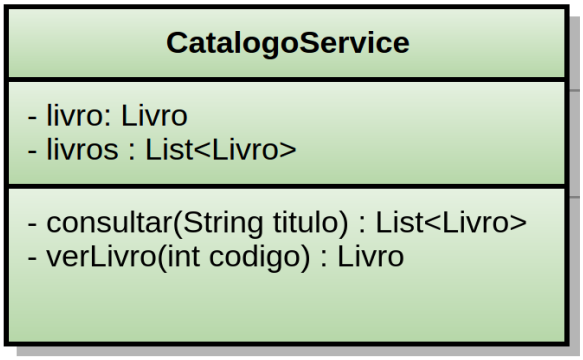
```
	private static final String OBTER_POR_ID_SQL = "SELECT AUTOR, TITULO,
COD_LIVRO, IMAGEM,"
			+ " PRECO, DESCRICAO FROM ESTOQUE WHERE COD_LIVRO = ?";

	private static final String CONSULTAR_SQL = "SELECT COD_LIVRO,
TITULO, AUTOR, PRECO,"
			+ " IMAGEM, DESCRICAO FROM ESTOQUE WHERE TITULO LIKE ?";

	@Override
	public Livro consultar(Integer codigo) {
		Livro livro = null;
		try (Connection conexao = FabricaConexao.getConexao();
				PreparedStatement consulta =
conexao.prepareStatement(OBTER_POR_ID_SQL);) {

			consulta.setInt(1, codigo);

			ResultSet resultado = consulta.executeQuery();

			if (resultado.next()) {
				livro = new Livro();
				livro.setAutor(resultado.getString("AUTOR"));
				livro.setCodigo(resultado.getInt("COD_LIVRO"));
				livro.setImagem(resultado.getString("IMAGEM"));
				livro.setPreco(resultado.getDouble("PRECO"));
				livro.setTitulo(resultado.getString("TITULO"));
				livro.setDescricao(resultado.getString("DESCRICAO"));
			}

			resultado.close();

		} catch (SQLException e) {
			LOG.severe(e.toString());
		}
		return livro;
	}

	public List<Livro> consultar(String titulo) {
		ArrayList<Livro> lista = new ArrayList<Livro>();
		try (Connection conexao = FabricaConexao.getConexao();
				PreparedStatement consulta =
conexao.prepareStatement(CONSULTAR_SQL);) {

			consulta.setString(1, "%" + titulo.toUpperCase() + "%");

			ResultSet resultado = consulta.executeQuery();

			while (resultado.next()) {
				Livro livro = new Livro();
				livro.setAutor(resultado.getString("AUTOR"));
```

```
                livro.setCodigo(resultado.getInt("COD_LIVRO"));
                livro.setImagem(resultado.getString("IMAGEM"));
                livro.setPreco(resultado.getDouble("PRECO"));
                livro.setTitulo(resultado.getString("TITULO"));
                livro.setDescricao(resultado.getString("DESCRICAO"));
                lista.add(livro);
            }
            resultado.close();

        } catch (SQLException e) {
            LOG.severe(e.toString());
        }
        return lista;
    }
    //Faça o restante do CRUD
    }
```

Você deve terminar de implementar o restantes do métodos definidos na Interface Dao generica.

Exercício 4 – Criando o Service e o Bean de consulta

1. Criado o Dao da aplicação, vamos fazer ela interagir diretamente com o nosso Managed Bean através de uma classe que daremos o nome de **CatalogoService.**

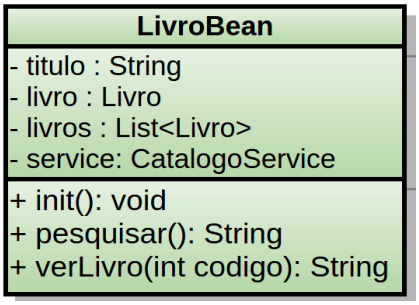



```
public List<Livro> consultar(String titulo){

      if(titulo == null){
            titulo = "";
      }
      System.out.println("Pesquisa: "+titulo);
      LivroDao dao = new LivroDao();
      livros = dao.consultar(titulo);

      if(livros.size() <= 0){
            System.out.println("Livro não localizado");
      }
      return livros;
}


public Livro verLivro(Integer codigo){

      LivroDao dao = new LivroDao();
```

```
        livro = dao.consultar(codigo);
        return livro;
    }
```

OBS: consultar(titulo) e consultar(codigo) são métodos do LivroDao, mas que buscam atributos diferentes (um busca um array de livros e outro busca somente um livro).

2. Nesse passo, vamos criar um managed bean chamado **LivroBean** e nele faremos um método **pesquisar()**, em que iremos receber uma string contendo a propriedade **titulo** através do formulário e consultar em nossa tabela **estoque** através da classe **LivroDao.** Segue o exemplo abaixo:




```
@ManagedBean

@SessionScoped

private static final String RESULTADO = "Resultado";

private static final String LIVRO = "Livro";


@PostConstruct

private void init(){
      service = new CatalogoService();
      System.out.println("LivroBean.init()");
}


public String pesquisar(){

      livros = null;
      livros = service.consultar(titulo);

      return RESULTADO;
}

public String verLivro(){

      livro = service.verLivro(codigo);

      return LIVRO;
}
```

OBS: “Resultado” e “Livro” nesse caso são o nome das páginas XHTML relacionadas. No caso, estamos fazendo uma navegação dinâmica implícita. Nos próximos passos iremos criar cada uma dessas páginas.

Exercício 5 – Desenvolvendo a página Inicial

1. Refaça os templates criados no **Laboratório de Templates**, assim como foi passado no laboratório. Códigos abaixo.

LayoutPadrao.xhtml

```
<h:head>

      <title><ui:insert name="titulo">Livraria</ui:insert></title>
      <h:outputStylesheet library="css" name="reset.css"/>
      <h:outputStylesheet library="css" name="bootstrap.css"/>
      <h:outputStylesheet library="css" name="estilo.css"/>
</h:head>
<h:body>
      <div class="page">
      <div class="header container">
            <ui:insert name="header">
                  <ui:include src="/WEB-INF/template/basicheader.xhtml" />
            </ui:insert>
      </div>
      <div class="content container">
            <ui:insert name="content">
                  <ui:include src="/WEB-INF/template/basiccontent.xhtml" />
            </ui:insert>
      </div>
      <div class="footer">
            <ui:insert name="footer">
                  <ui:include src="/WEB-INF/template/basicfooter.xhtml" />
            </ui:insert>
      </div>
</div>
</h:body>
```

basicheader.xhtml

```
 <body>

<ui:composition>
<header>
      <h:link>
            <h:graphicImage library="imagens" name="Logo-Triway.png"
                  id="logo-livraria"/>
      </h:link>

      <nav class="menu-opcoes">
            <ul>
                  <h:form>
                        <li><h:link value="Carrinho"/></li>
                        <li><h:link value="Cadastro" /></li>
                        <li><h:link value="Login" /></li>
                  </h:form>
            </ul>
      </nav>
</header>
</ui:composition>
</body>
```

basicheader.xhtml

```
<body>

<ui:composition>
      <footer>
            <div class="container">
                  <h:link>
                        <h:graphicImage library="imagens" name="Logo-Triway.png"
                              id="logo-rodape"/>
                  </h:link>
                  <ul class="social">
                        <li><a href="http://facebook.com/">Facebook</a></li>
                        <li><a href="http://twitter.com/">Twitter</a></li>
                        <li><a href="htto://plus.google.com/">Google+</a></li>
                  </ul>
            </div>
      </footer>
</ui:composition>
</body>
```

2. Crie na pasta **webapp** o arquivo **Inicio.xhtml.** Essa será a página inicial da sua livraria virtual. Nela, vamos criar um menu de departamentos e um formulário de busca, onde iremos consultar os livros em estoque. Segue exemplo do código:

```
<h:body>

<ui:composition template="/WEB-INF/template/LayoutPadrao.xhtml">
      <ui:define name="content">
            <div class="container">
                  <div class="destaque">
                        <section class="busca">
                        <h2>Busca</h2>
                        <h:form>
                              <h:inputText id="search"
                                    value="#{livroBean.titulo}" size="14”/>
                              <h:commandButton icon="ui-icon-search"
                                    action="#{livroBean.pesquisar}" />
                        </h:form>
                        </section>

                        <section class="menu-departamentos">
                        <h2>Departamentos</h2>
                        <nav>
                        <h:form>
                        <ul>
                              <li><h:commandLink value="Livros" />
                                    <ul>
                                    <li><h:commandLink value="Auto-Ajuda" /></li>
                                    <li><h:commandLink value="Bibliografias"/></li>
                                    <li><h:commandLink value="Teen"/></li>
                                    <li><h:commandLink value="Romance" /></li>
                                    <li><h:commandLink value="HQs" /></li>
                                    </ul>
                              </li>
                              <li><h:commandLink value="Filmes" /></li>
                              <li><h:commandLink value="Games" /></li>
                              <li><h:commandLink value="Musica" /></li>
                              <li><h:commandLink value="Acessorios" /></li>
                        </ul>
```

```
                        </h:form>

                        </nav>
                        </section>
                    </div>
                </div>
        </ui:define>
</ui:composition>
</h:body>
```

CSS:

```
.busca,
.menu-departamentos{
        background-color: #dcdcdc;
        font-weight: bold;
        text-transform: uppercase;
        margin-right: 10px;
        width: 230px;
}

.busca h2,
.busca form,
.menu-departamentos h2{
        font-size:15px;
        font-weight:bold;
        margin: 10px;
}

.menu-departamentos li{
        background-color: white;
        margin-bottom: 1px;
        padding: 5px 10px;
}

.menu-departamentos a{
        color: #333;
        text-decoration: none;
}

.busca input{
        vertical-align: middle;
}

.busca input[type=search]{
        width: 170px;
}

.busca,
.menu-departamentos{
        float:left;
}

.menu-departamentos li ul{
        display: none;
}

.menu-departamentos li:HOVER ul{
        display: block;
}
```

```
.menu-departamentos ul ul li{
	background-color: #dcdcdc;
}

.menu-departamentos li li a:BEFORE{
	content: '\261E ';
	padding-right: 3px;
}

.menu-departamentos{
	clear: left;
	margin-top: 2px;
	padding-bottom: 10px;
}
```

Pronto, temos nossa página inicial, porem ela ainda não está funcional, precisamos da página Resultado.xhtml para onde o PesquisaBean vai redirecionar o resultado da consulta.

Exercício 6 – Desenvolvendo a página de Resultados

1. Na mesma pasta webapp, crie agora o arquivo **Resultado.xhtml** e usando os templates, faça com que mostre uma lista com todos os livros encontrados na nossa busca da página inicial. A section de **busca** e **departamentos** não é um template, mas pode ser usado como um. Faça com que esses elementos apareçam no **Resultado.xhtml**.

```
<ui:composition template="/WEB-INF/template/layout.xhtml">

<ui:define name="content">
 <div class="container">

 <div class="row">

	<div class="col-md-3">
		BUSCA E DEPARTAMENTOS
	</div>
	<div class="col-md-9">
		<ui:repeat value="#{livroBean.livros}" var="livro">
		<div class="panel">
			<div class="row">
			<div class="col-md-3">
				<img src="#{livro.imagem}" id="img-livro"/>
			</div>
			<div class="col-md-8 descr-livro">
				<h3 class="titulo">#{livro.titulo}</h3>
				<h4 class="pull-right preco">R$ #{livro.preco}</h4>
				<h4 class="autor">#{livro.autor}</h4>
				<p class="descricao">#{livro.descricao}</p>
				<h:form>
					<h:inputHidden id="codigo" value="#{livro.codigo}"/>
					<h:commandButton value="Ver livro"
					action="#{livroBean.verLivro()}"/>
				</h:form>
			</div>
			</div>
		</div>
	</ui:repeat>
	</div>
```

```
</div>
</div>
```

CSS:

```
.descricao{
	margin: 10px 0;
	font-weight:100;
}
.preco{
	color: #204a87;
	font-weight: bold;
	padding-right: 10px;
}
.autor{
	font-weight: bold;
	color: gray;
}
.titulo{
	font-weight: bold;
	font-size: 23px;

	padding: 5px 0;
	margin-bottom: 5px;
	color: #3465a4;
}
#img-livro{
	width: 150px;
	margin: 2px;
	padding: 5px 1px;
	float: left;
}
.panel-body{
	margin:5px;
}
.panel{
	margin: 15px;
	position:relative;
}
```

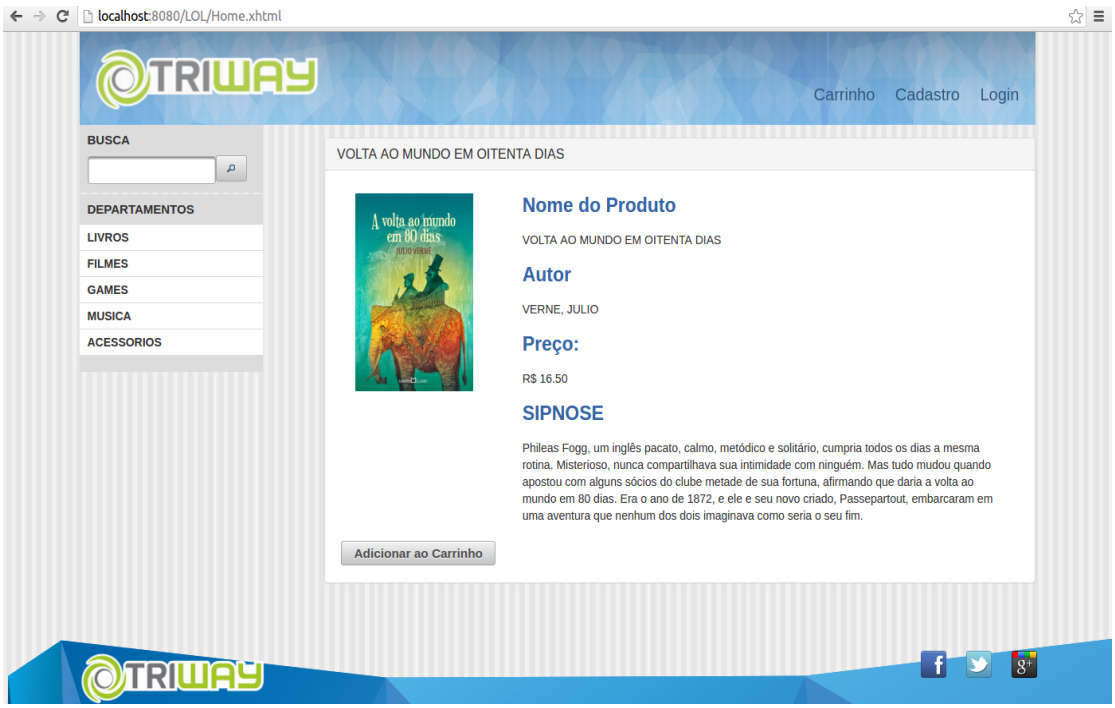
Agora sim, seu formulário de busca funcionará. Acesse a página Inicio.xhtml e teste o formulário de busca, veja o resultado.

Exercício 7 – Desenvolvendo a página de Livros

1. Crie agora o arquivo **Livro.xhtml** e faça com que ele mostre para nós com detalhes o livro escolhido pelo usuário, na página de resultados. Use os templates para completar o design da página.

```
<ui:composition template="/WEB-INF/template/layout.xhtml">

<ui:define name="content">

<div class="container">

<div class="row">

   <div class="col-md-3">
         BUSCA E DEPARTAMENTOS
    </div>
<div class="col-md-9">

   <div class="row">
         <div class="panel panel-default">
         <div class="panel-heading">
               <h2 class="panel-title">#{livroBean.livro.titulo}</h2>
         </div>
         <div class="panel-body">
               <div class="col-md-3">
                     <img src="#{livroBean.livro.imagem }" id="img-livro" />
               </div>
               <div class="col-md-9">
               <dl>
                     <dt class="titulo">Nome do Produto</dt>
                     <dd class="descricao">#{livroBean.livro.titulo}</dd>
                     <dt class="titulo">Autor</dt>
                     <dd class="descricao">#{livroBean.livro.autor}</dd>
                     <dt class="titulo">Preço:</dt>
                     <dd class="descricao">R$ #{livroBean.livro.preco}</dd>
                     <dt class="titulo">SIPNOSE</dt>
                     <dd class="descricao">#{livroBean.livro.descricao}</dd>
               </dl>
               </div>
               <h:form>
                     <h:inputHidden id="codigo"
                           value="#{pesquisaBean.livro.codigo}" />
                     <p:commandButton value="Adicionar ao Carrinho"/>
               </h:form>
         </div>
         </div>
   </div>
</div>
```

Página criada, teste novamente o seu formulário de busca porem, ao chegar na página de resultados, clique em **Ver Livro** e será direcionado para a página Livro.xhtml, que deverá se parecer com o exemplo abaixo. Adicionamos o botão **Adicionar ao Carrinho** mas sem ação, pois será usada futuramente, no workshop de Frameworks.



Exercício 8 – Cadastrando novos Livros na página

1. Vamos agora fazer uma página para acrescentar livros ao estoque da sua livraria. Para isso, iremos voltar ao **LivroDao** e implementar o método salvar() no crud do nosso dao. Acrescente ao seu código as passagens abaixo, tentando organizá-las o máximo possível.

```
private static final String SALVAR_SQL = "INSERT INTO ESTOQUE (TITULO, AUTOR,
 PRECO, IMAGEM, DESCRICAO) VALUES (?,?,?,?,?)";

    …

public void salvar(Livro livro) {

    try(Connection conexao = FabricaConexao.getConexao();
          PreparedStatement consulta = conexao.prepareStatement(SALVAR_SQL);){
          consulta.setString(1, livro.getTitulo());
          consulta.setString(2, livro.getAutor());
          consulta.setDouble(3, livro.getPreco());
          consulta.setString(4, livro.getImagem());
          consulta.setString(5, livro.getDescricao());
          consulta.execute();

    }
    catch(SQLException e){
          LOG.severe(e.toString());
    }

}
```

2. Volte ao **CatalogoService** e nele, crie o método **cadastroLivro()**, assim como mostrado abaixo.

```
public void cadastroLivro(Livro livro){

	LivroDao dao = new LivroDao();
	dao.salvar(livro);
}
```

3. Feito o Service, vamos ao Bean. Vá em **LivroBean** e nele acrescente o método **cadastrar().** Porem, para fazer esse cadastro, precisaremos receber as informações de titulo, autor, preço do livro. Então, acrescente também à classe LivroBean os atributos: titulo(String), autor(String), preco(Double), imagem(String), e descricao(String). Segue o código abaixo. Não se esqueça dos **getters e setters.**

```
public void cadastrar(){

	Livro livro = new Livro();
	livro.setTitulo(titulo.toUpperCase());
	livro.setAutor(autor.toUpperCase());
	livro.setPreco(preco);
	livro.setImagem("imagens/");
	livro.setDescricao(descricao);
	System.out.println("LivroBean.cadastrar()");
	service.cadastroLivro(livro);
}
```

4. Para fazer o teste, vamos criar uma página **InserirLivro.xhtml** e nela vamos criar um painel onde o admin irá cadastrar o livro. Segue o código e CSS abaixo.

```
<ui:composition template="WEB-INF/template/layout.xhtml">

	<ui:define name="content">
		<div class="container">
			<div class="panel panel-primary painel-cadastro">
			<div class="panel-heading">Cadastro</div>
			<div  class="panel-body">
				<h:panelGrid>
				<h:form id="lvr">
					<div class="form-group">
						<h:outputLabel value="Titulo" for="titulo"/>
						<h:inputText id="titulo" required="true"
					class="form-control" value="#{livroBean.titulo}"/>
					</div>
					<div class="form-group">
						<h:outputLabel value="Autor"/>
						<h:inputText id="autor" required="true"
					class="form-control" value="#{livroBean.autor}"/>
					</div>
					<div class="form-group">
						<h:outputLabel value="Preço"/>
						<h:inputText id="preco" required="true"
					class="form-control" value="#{livroBean.preco}" />
					</div>
					<div class="form-group">
						<h:outputLabel value="Descrição"/>
```

```
                                    <h:inputText id="descr"
                                    value="#{livroBean.descricao}"
                                    class="form-control"/>
                              </div>
                              <h:commandButton value="Adicionar Livro"
                                    action="#{livroBean.cadastrar()}"/>
                        </h:form>
                        </h:panelGrid>
                  </div>
                  </div>
                  </div>
      </ui:define>
</ui:composition>
```

Abra a página e faça um teste, veja se consegue inserir um livro novo à sua tabela de estoque.

5. Repare que, ao cadastrar um livro novo, ainda não estamos colocando a sua imagem de capa. Para isso, iremos usar a tag `h:inputFile`. Volte ao **InserirLivro.xhtml** e adicione a tag assim como mostrado abaixo.

```
<h:panelGrid>

      <h:form enctype="multipart/form-data" id="img-livro">
            <h:inputFile id="file"/>
      </h:form>
```

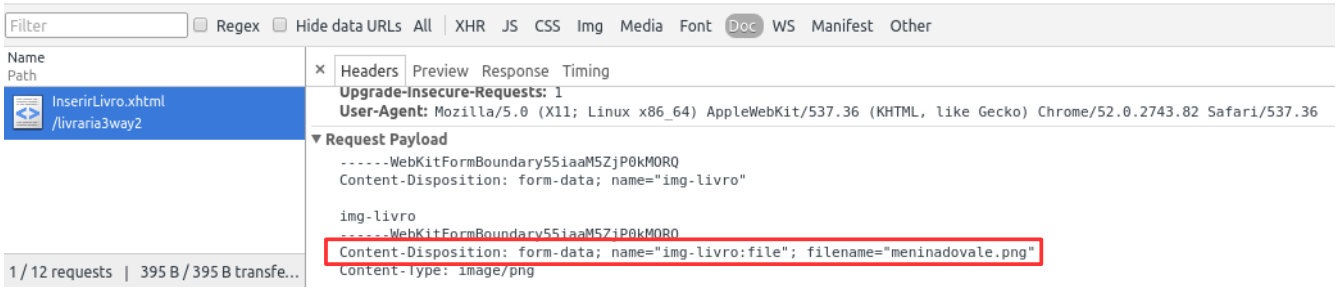
6. Mas para essa tag funcionar, devemos montar um método no nosso **LivroBean**. Volte ao seu bean e crie o atributo **file** o método **upload(),** assim como mostrado no código abaixo.

```
import javax.servlet.http.Part;

private Part file;

public void upload(){

}
```

7. Retorne ao **InserirLivro.xhtml** e altere o form conforme abaixo. Em seguida, desenvolva uma página simples **Sucesso.xhtml** para termos uma visão do resultado.

```
<h:panelGrid>

      <h:form enctype="multipart/form-data" id="img-livro">
            <h:inputFile id="file" value="#{livroBean.file}">
                  <f:ajax listener="#{livroBean.upload()}"/>
            </h:inputFile>
      </h:form>
```

8. Teste novamente o inputFile, mas agora abra a *ferramenta do desenvolvedor* do navegador usado apertando f12, vá para a tag network e tente inserir uma imagem no input, assim como feito abaixo. Veja o caminho onde a imagem está salvando.



9. O caminho marcado em vermelho será onde iremos pegar o nome do nosso arquivo. Agora, vamos criar o método getFilename() para filtrar esse nome do arquivo que chega para a máquina. Observe atentamente o código abaixo, fazendo as alterações necessárias caso a imagem acima seja diferente da encontrada em seu navegador.

```
private static String getFilename(Part part){
      for(String cd : part.getHeader("content-disposition").split(";")){
            if(cd.trim().startsWith("filename")){
               String filename = cd.substring(cd.indexOf('=')+1).trim().replace("\"","");
               return filename.substring(filename.lastIndexOf('/')+1);
            }
      }
      return null;
}
```

10. Pronto, conseguimos filtrar o nome do arquivo. Agora, vamos salvá-lo diretamente na pasta de imagens usada no projeto. Procure e copie o caminho da sua pasta de imagens do projeto ou crie um link da sua pasta para um caminho mais facil. Volte ao método **upload()** e escreva o seguinte código. **NÃO COPIE O CÓDIGO ABAIXO.**

```
  public void upload(){
      try{
            //file.write para salvar a imagem na pasta desejada
            file.write("/home/fabrica-3way/link-imagens/"+getFilename(file));
            System.out.println(getFilename(file));
      } catch(IOException e){
            e.printStackTrace();
      }
  }
```

Altere também em cadastrar(), o cadastro de imagem e faça com que ao terminar o cadastro, o admin seja mandado a uma página de sucesso simples.

```
livro.setImagem("imagens/"+getFilename(file));
return SUCESSO ;
```

11. Agora, faça um teste adicionando um livro qualquer à sua livraria, escolha a imagem e as informações. Veja se esse livro foi cadastrado acessando a sua página de pesquisa de livros.