Módulo 7

No módulo anterior aprendemos o mecanismo de controle de acesso provido pelo Demoiselle para facilitar a implementação de autenticadores e autorizadores. Experimentamos a anotação *@RequiredPermission* e conhecemos a interface que reúne funcionalidades de segurança do Framework, a *SecurityContext*.

Nossa aplicação já atende aos requisitos básicos da arquitetura em camadas: entidades que transportam dados entre as camadas e separação entre regras de negócio e persistência dos dados. Nos preocupamos com *log*, *resource bundle*, tratamento de exceções, persistência, controle transacional e arquitetura. O que faremos neste módulo então? Vamos dar uma "cara" web ao nosso projeto utilizando JSF.

Arquitetura

Até agora validamos as implementações com casos de teste utilizando JUnit. A partir deste momento focaremos na camada de apresentação, e não mais no negócio ou persistência. A camada de apresentação é responsável por exibir as funcionalidades implementadas nas camadas internas da aplicação (negócio e persistência) para o usuário. Uma discussão muito polêmica é: quem é o usuário de nossa aplicação? Depende!

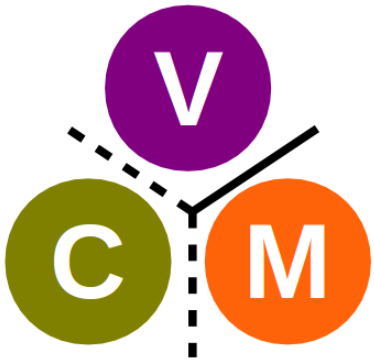
Poderíamos desenvolver aplicações destinadas a programadores ou operadores técnicos, neste caso criaríamos bibliotecas de funções ou programas de linha de comando. Outra hipótese seria disponibilizar serviços *web* (Web Services) para serem acessados por outros sistemas (usuários dos serviços) através de protocolos específicos. Contudo, na maioria das aplicações corporativas, os usuários não tem intimidade com programação e interagem com o sistema por meio de telas. Em todos os casos estamos falando da camada de apresentação.

Por questões de organização, recomenda-se que nenhuma camada seja burlada. Em termos práticos, a camada de apresentação acessa a camada de negócio que por sua vez acessa a de persistência. Perceba que a comunicação entre elas ocorre em apenas um sentido. Isto significa que a camada de persistência não deve acessar as camadas de negócio ou apresentação, e assim por diante.

MVC

Nós vamos utilizar a especificação JSF (JavaServer Faces) no nosso projeto. O JSF foi criado com o intuito de facilitar a integração dos elementos *web* com o mundo Java. Veremos isto na prática! Assim como diversas outras tecnologias para a camada de apresentação, o JSF implementa o padrão MVC (*Model-View-Controller*). Você sabe exatamente qual o objetivo do MVC? Para dificultar sua vida? Não mesmo!

Antes da popularização do MVC, as aplicações *web* pareciam espaguete: elementos HTML misturados com lógica de programação e SQL. Quem já teve o desprazer de trabalhar num ambiente hostil como este sabe exatamente o que estou falando. O MVC surgiu para pôr ordem nesta bagunça, separando o que é visual (*View*) e o que é lógica do negócio (*Model*). Separar o visual da lógica de negócio não foi suficiente, ainda havia uma costura a fazer. Foi daí que surgiu o controlador (*Controller*) com a responsabilidade de intermediar a comunicação entre os outros dois elementos.



Na prática, o padrão MVC recomenda que o visual, também conhecido como visão, não acesse (linha contínua) diretamente as bases lógicas da aplicação, também conhecida como modelo. O controlador funciona como uma espécie de pombo correio e implementa o código-cola. Agora sim tem-se as responsabilidades bem definidas: visual (*View*), lógica (*Model*) e cola (*Controller*).

Utilizamos na nossa aplicação os estereótipos *@BusinessController* e *@PersistenceController*. Seriam eles controladores MVC? Não! O sufixo *Contoller* utilizado pelo Demoiselle para representar os controladores das camadas nada tem a ver com o MVC. Aliás, é muito comum confundirem MVC com arquitetura em camadas. Não seja um deles!

Estrutura

Abra o *pom.xml* e observe a sessão ***parent***. Nos módulos anteriores utilizamos o *demoiselle-minimal-parent*. Atualizaremos o nosso projeto para herdar do *demoiselle-jsf-parent*, que contém uma série de definições prontas para aplicações JSF, tais como bibliotecas e configurações de empacotamento do projeto.

1. Na sessão ***parent*** do *pom.xml*, modifique a entrada *demoiselle-minimal-parent* da para `demoiselle-jsf-parent`.
2. Na sessão ***packaging***, modifique a entrada *jar* para *war*.
3. Na sessão ***dependencies***, para os artefatos EclipseLink , HSQLDB e SLF4J-API defina o escopo para ***test***, pois o servidor JEE 6, no caso o Jboss que usaremos na sequencia, já provê dependências equivalentes.
4. Execute o comando no Menu do Eclipse (selecionando o projeto clicando botão direito do mouse): *Maven / Update Project Configuration*. (marque a opção: Force Update of Snapshots/Releases)
5. Crie a estrutura de pasta *src/main/webapp/WEB-INF.*
6. Crie o arquivo ***web.xml*** vazio na pasta *WEB-INF*.

As modificações no *pom.xml* serão as seguintes:

```
...
<packaging>war</packaging>
…
<parent>
   <groupId>br.gov.frameworkdemoiselle</groupId>
   <artifactId>demoiselle-jsf-parent</artifactId>
   <version>2.3.0</version>
</parent>
...
<dependencies>
   <dependency>
      <groupId>br.gov.frameworkdemoiselle</groupId>
      <artifactId>demoiselle-jpa</artifactId>
   </dependency>
<dependency>
      <groupId>org.eclipse.persistence</groupId>
      <artifactId>eclipselink</artifactId>
      <scope>test</scope>
   </dependency>
<dependency>
      <groupId>hsqldb</groupId>
      <artifactId>hsqldb</artifactId>
      <scope>test</scope>
   </dependency>
<dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <scope>test</scope>
   </dependency>
</dependencies>
...
```

Nossa aplicação está ficando cada vez mais profissional. Informamos ao Maven que nosso projeto deve ser empacotado como *war*, ou seja, uma aplicação Java *web*. Ao atualizar as configurações, o *plug-in* Maven modificou os artefatos específicos do Eclipse para identificar a aplicação como um projeto *web*.

Deixe o *web.xml* assim:

```
<?xml version="1.0"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
</web-app>
```

Clique com o botão direito no projeto e acesse *Maven / Update Project Configuration*. (marque a opção: Force Update of Snapshots/Releases)

Graças às facilidades da especificação JEE 6, ambos arquivos não precisam de configurações adicionais. Deixe-os assim mesmo: cabeçalhos XML definidos, porém sem conteúdo.

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-7-video-1

Olá mundo!

Que tal fazermos um ***Hello World*** para ver nossa aplicação web rodando? Vamos lá!

- Crie o arquivo ***index.html*** na pasta ***src/main/webapp***;
- Abra o arquivo e escreva "***Olá mundo!***";
- Acesse a pasta ***src/main/resources/META-INF***;
- Crie o arquivo persistence.xml com o seguinte conteúdo:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
      <persistence-unit name="inscricao-ds" transaction-type="RESOURCE_LOCAL">
            <non-jta-data-source>java:jboss/datasources/ExampleDS</non-jta-data-
source>
<class>br.gov.serpro.inscricao.entity.Aluno</class>
            <properties>
                  <property name="hibernate.show_sql" value="true" />
                  <property name="hibernate.format_sql" value="false" />
                  <property name="hibernate.hbm2ddl.auto" value="create-drop" />
            </properties>
      </persistence-unit>

</persistence>
```

- Copie o arquivo **beans.xml** do diretório **/src/test/resources/META-INF** para o diretório **/src/main/webapp/WEB-INF/**
- No Eclipse, Arraste o seu projeto para o elemento **JBoss (confira a versão recomendada pela comunidade)** da aba **Servers;**
- Clique com o botão direito em **JBoss** e selecione **Start**;
- Aguarde o servidor iniciar;
- Acesse o endereço **http://localhost:8080/inscricao**.

Ao abrir o navegador, deverá aparecer a frase: “Olá Mundo”.

Ao invés do HTML puro, vamos utilizar recursos JSF. Crie o arquivo *turma.xhtml* na pasta *src/main/webapp* com o seguinte conteúdo:

```
<html xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html">
        Olá Mundo!
</html>
```

Altere o arquivo /src/main/webapp/WEB-INF/web.xml , conforme mostrado abaixo:

```
<?xml version="1.0"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
```

```
version="3.0">

	<servlet>
		<servlet-name>Faces Servlet</servlet-name>
		<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>Faces Servlet</servlet-name>
		<url-pattern>*.jsf</url-pattern>
	</servlet-mapping>

</web-app>
```

Acesse o endereço **http://localhost:8080/inscricao/turma.jsf** e veja o resultado.

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-7-video-2

Listagem

Criaremos agora uma tela que exibirá os alunos matriculados na turma. Faça o seguinte:

1. Crie a classe TurmaMB no pacote br.gov.serpro.inscricao.view.
2. Anote-a com @ViewController.
3. Acrescente o atributo @Inject private TurmaBC bc.
4. Crie o método ***public List getAlunosMatriculados()*** que acessa o BC e retorna os alunos matriculados.

A classe ***TurmaMB*** ficará assim:

```
@ViewController
public class TurmaMB {

	@Inject
	private TurmaBC bc;

	public List<Aluno> getAlunosMatriculados(){
		return bc.obterAlunosMatriculados();
	}
}
```

Partiremos agora para a página. Abra o arquivo *turma.xhtml* e acrescente o seguinte trecho de código que utiliza componentes JSF:

```
<html xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html">
	<h:body>
		<h:dataTable id="lista" var="aluno" value="#{turmaMB.alunosMatriculados}">
			<h:column>
				<h:outputText value="#{aluno.matricula}" />
			</h:column>
			<h:column>
				<h:outputText value="#{aluno.nome}" />
			</h:column>
		</h:dataTable>
	</h:body>
</html>
```

Observe que cada cada elemento da listagem é associado à variável “aluno”, que é utilizada para exibir os dados à cada iteração.

Re-inicie o servidor e acesse o endereço http://localhost:8080/inscricao/turma.jsf. Como nenhum aluno está matriculado, nada será listado mas será possível verificar no console do Eclipse a execução do método.

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-7-video-3

Matrícula

Como matricular o aluno? Assim:

1. Abra a classe ***TurmaMB***.
2. Acrescente o atributo *private* ***String nomeAluno***.
3. Gere os métodos *get* e *set* para o atributo *nomeAluno*.
4. Crie o método *public void* ***matricular()*** sem parâmetros que acessa o BC para efetivar a matrícula do aluno.

As modificações na classe ***TurmaMB*** serão estas:

```
@ViewController
public class TurmaMB {

	...

	private String nomeAluno;

	...

	public String getNomeAluno() {
		return nomeAluno;
	}

	public void setNomeAluno(String nomeAluno) {
		this.nomeAluno = nomeAluno;
	}

	public void matricular(){
		bc.matricular(new Aluno(nomeAluno));
	}
}
```

Abra o arquivo *turma.xhtml* e acrescente o seguinte trecho de código antes da tabela:

```
...
<h:form>
	Nome do aluno
	<h:inputText id="nomeAluno" value="#{turmaMB.nomeAluno}"/>
	<h:commandButton value="Matricular" action="#{turmaMB.matricular}" />
</h:form>
...
```

Utilizamos mais alguns elementos JSF. O *inputText* mantém o elemento de tela sincronizado com o atributo *nomeAluno* da classe *TurmaMB*, onde os valores serão definidos e obtidos através dos métodos assessores (*get* e *set*). A sincronização ocorrerá entre o acionamento do *commandButton* e a chamada do método *action*.

Acesse o endereço http://localhost:8080/inscricao/turma.jsf e experimente incluir alguns registros. Veja o que ocorrerá!

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-7-video-4

Retrospectiva

Conhecemos a camada de apresentação e vimos cada um dos elementos do MVC. Aprendemos que arquitetura em camadas não é a mesma coisa que MVC. Refatoramos nosso projeto para adequar-se aos padrões de uma aplicação *web*. Utilizamos o POM *demoiselle-jsf-parent* e criamos os arquivos de configuração *web.xml* e *faces-config.xml*. Construímos a classe ***TurmaMB***: um controlador MVC integrante da camada de apresentação. Listamos na tela e matriculamos alunos.

No próximo módulo, veremos recursos do Demoiselle para manipulação de mensagens com o usuário e criaremos aplicações *web* de forma rápida a partir das facilidades do Framework.