Spago Tutorial

Authors

Gianfranco Boccalon
Daniela Butano
Gabriele Ruffatti

Index

VERSIONS HISTORY 3

DOCUMENT GOAL 3

REFERENCES 3

1 THE SAMPLE 4

2 DEPENDENCIES 4

3 CREATE A NEW PROJECT 5

4 PROJECT CONFIGURATION 7

4.1 LIBRARIES 7
4.2 CONFIGURATION FILES 7
4.3 WEB.XML 8
4.4 TAG LIBRARIES 8
4.5 STILESHEETS, IMAGES AND JSP 8

5 ARCHITECTURAL DESIGN 9

6 DEVELOPMENT 10

6.1 ACTION.XML 10
6.2 LOGIN JSP PAGE 10
6.3 LOGIN ACTION 14
6.4 USERS CONFIGURATION FILES 15
6.5 MULTI-LANGUAGE CONFIGURATION 15
6.6 PRESENTATION.XML FILE 16
6.7 PUBLISHERS.XML FILE 16
6.8 WELCOME JSP PAGE 17

7 DEPLOY AND TEST 18

Versions History

Version/Release n° :	1.0	**Data Version/Release** :	November, 19th 2004
Description:	First release (english version)		
Version/Release n° :	1.1	**Data Version/Release** :	December, 13th 2004
Description:	Changed in the *master.xml* file the reference to *data_access.xml*		

Document Goal

The goal of this tutorial is to provide you with a simple introduction on configurating an applications implemented with Spago. We will use the template *Spago.war* released in *startup-sample.zip*

References

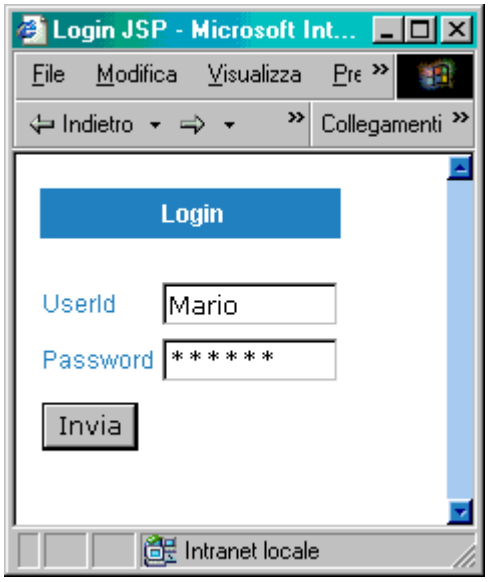
You can find more informations about Spago framework in the following documents (now in Italian, as soon as possible in English version) at: http://spago.eng.it/docs_it/documentation/index.html :

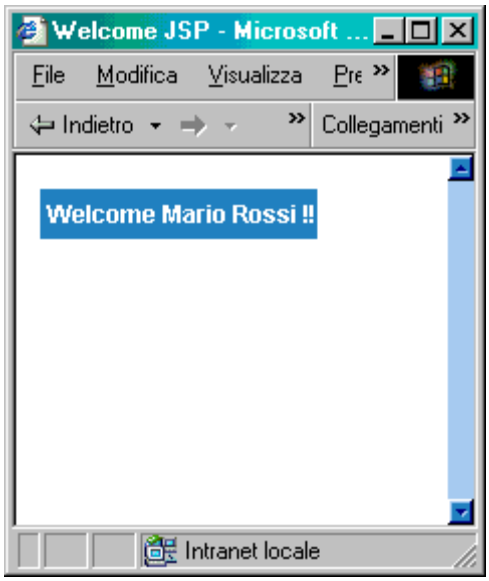
- [1] *Spago Overview*
- [2] *Spago User Guide.*

1 The sample

The following sample concerns a simple application composed by two JSP pages: login and welcome.

Users are recorded in an XML configuration file; you will access it through Spago.





2 Dependencies

The following is a list of software dependencies for the sample you need to download and extract to your system:

- **JDK 1.4.x** download from *http://java.sun.com/*
- **Tomcat 5.0.28** download from *http://jakarta.apache.org/*
- **Eclipse 3.0** download from *http://www.eclipse.org/*
- **Lomboz 3.0.1** download from *www.objectlearn.com*

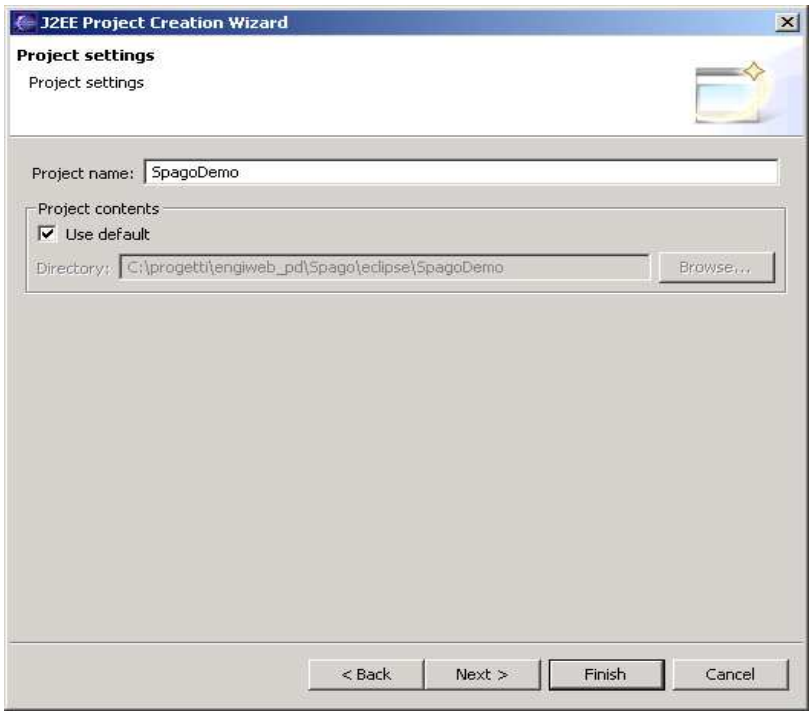
You have to unzip/install all of the above.

You need the Spago version distributed in *http://sourceforge.net/projects/spago*.

For the sample you'll use the project template *Spago.war* distributed in *http://sourceforge.net/project/showfiles.php?group_id=119426* in startup-sample.zip.

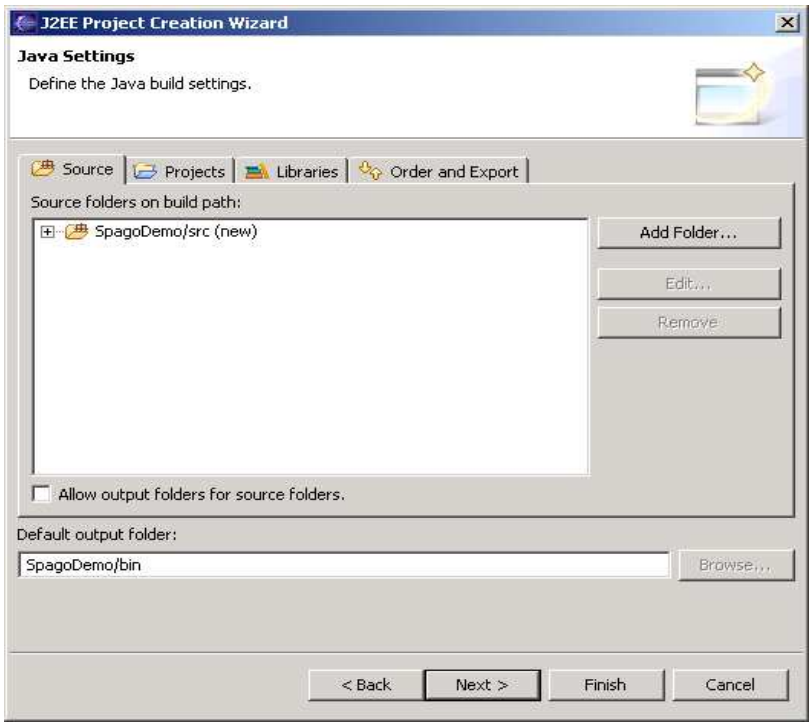
3 Create a new project

Start *Eclipse*, then create a new project for the sample application selecting *New->Lomboz J2EE Project*; the wizard will be the following:



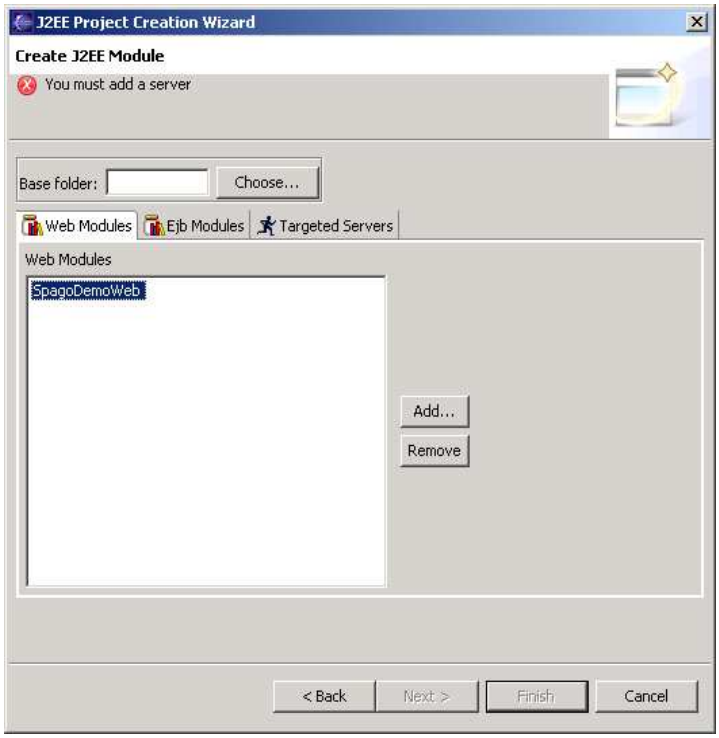
For the project name use "*SpagoDemo*".

Use the follows settings in "*Java settings*":

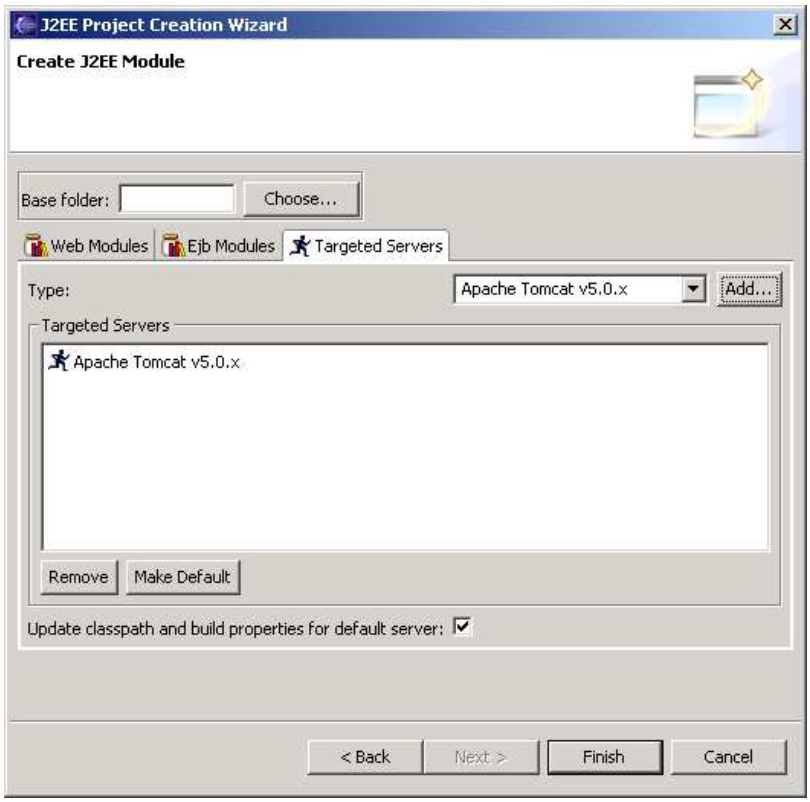


Set the default output folder as "***SpagoDemo/bin***".

Press the *Next* button and add a new web module *SpagoDemoWeb*.



Add *Tomcat5.0.x* as application server.



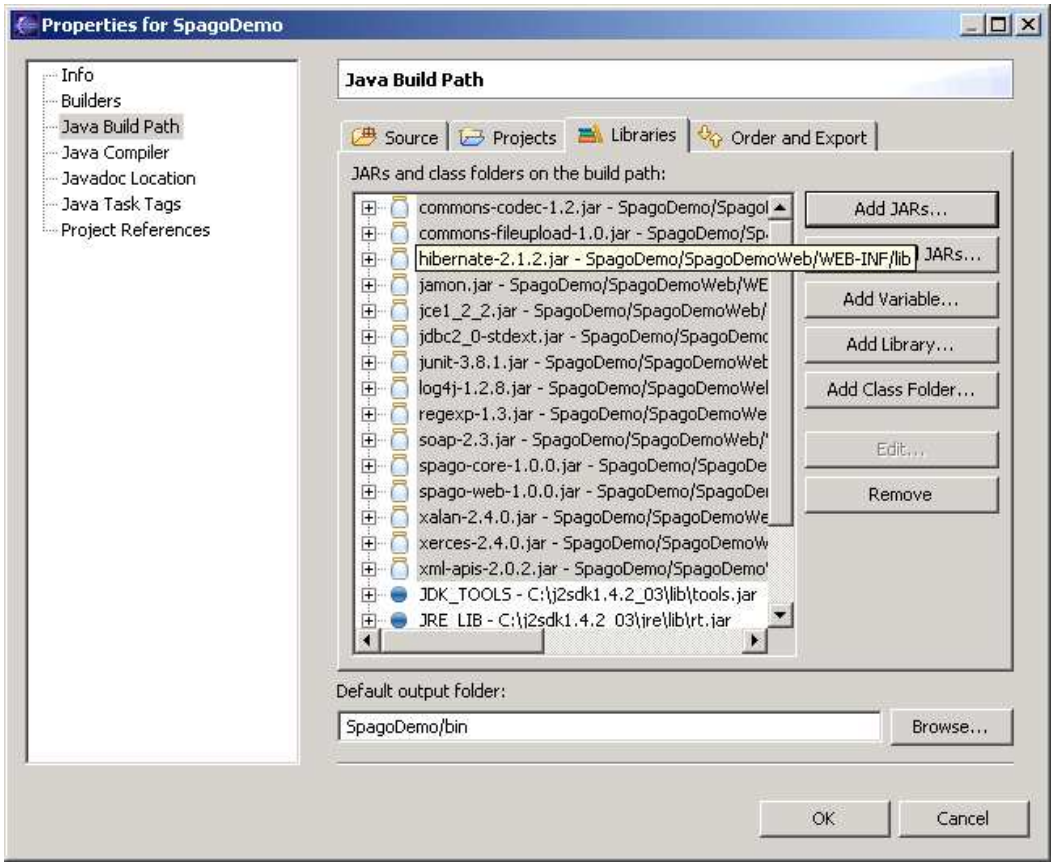
4 Project configuration

Now, we will show you the steps need to configure the new project in order to use Spago.

At the first, you must unzip *Spago.war* in *SpagoDemoWeb* folder. In this way, all configuration files, all external libreries and all Spago libraries, tags libraries, stylesheets, pages handlers of errors and exceptions are ready.

4.1 Libraries

First of all, you have to configure the project in order to use the libraries. Go to project properties, index "*Java Build Path*", tab *Libraries* and add all libraries in *SpagoDemoWeb/WEB-INF/lib*.



4.2 Configuration files

Spago needs the XML configuration files distibuted with Spago in the folder *WEB-INF/conf/spago*; now this files are in *WEB-INF/conf/spago* directory of your project.

4.3 Web.xml

The *web.xml* file contains the correct configurations for Spago's HTTP adapter and Spago's tag libraries.

If you want to save configuration files in an external directory, you must remove comments in the following section and modify the value of the *AF_ROOT_PATH* parameter with the directory path:

```
<init-param>
 <param-name>AF_ROOT_PATH</param-name>
 <param-value>C:\Progetti\eclipse\SpagoDemo\SpagoDemoWeb</param-value>
</init-param>
```

4.4 Tag Libraries

Spago provides you a tag library defined in the file "*\WEB-INF\tld**spago.tld***".

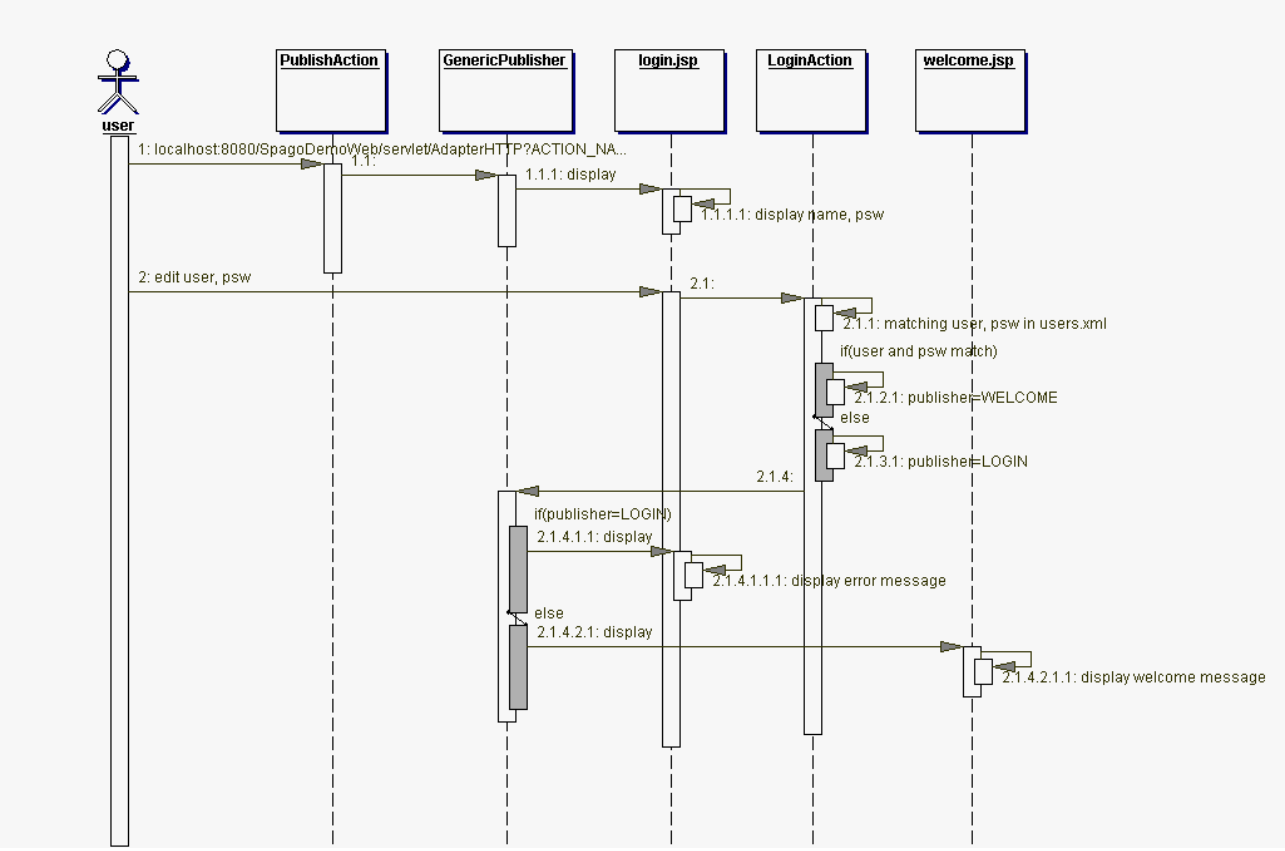
4.5 Stilesheets, images and JSP

You can automatically generate lists and detail forms using stylesheets and images defined in "*\WEB-INF\css\spago*" and "*\WEB-INF\img\spago*".

5 Architectural Design

The sample uses the action dispatching, a way of coding where a business object corresponds to a service request (another way is module dispatching: se documentation in References)

Here is the architectural schema for the application.



6 Development

The following are the steps to follow for the applicative development.

6.1 ACTION.XML

The sample uses two action:

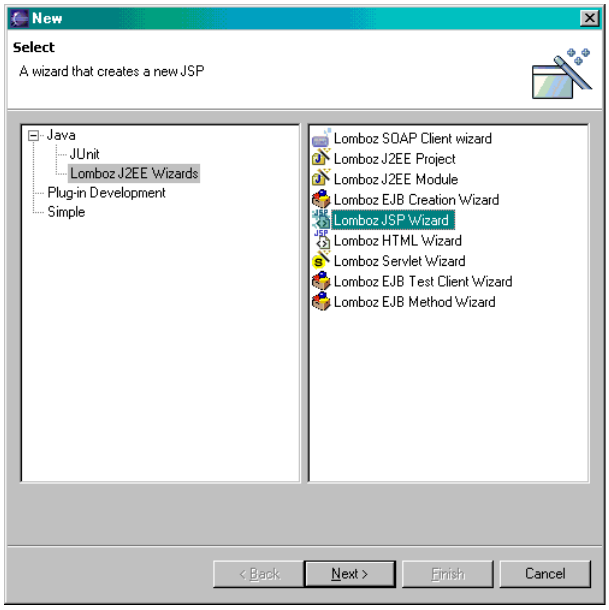
- *Login*: this action performs user authentication.
- *Publish*: Spago provides you this action to publish the login jsp page.

The "*SpagoDemo\SpagoDemoWeb\WEB-INF\conf\spago**actions.xml***" file structure should look like this:

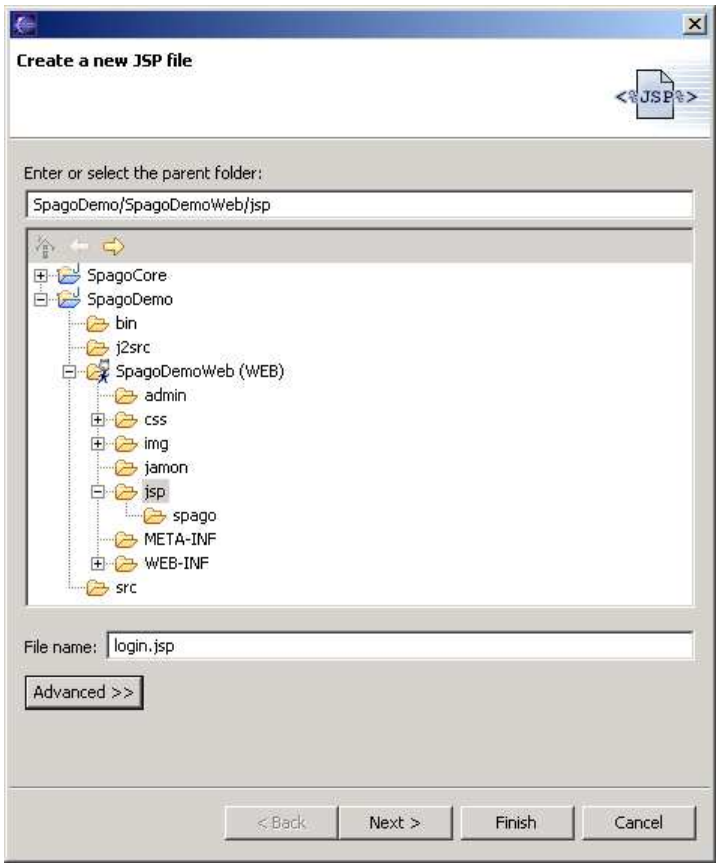
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ACTIONS>
      <ACTION name="LOGIN"
            class="it.eng.spago.demo.action.LoginAction"
            scope="REQUEST"/>
      <ACTION name="PUBLISH"
            class="it.eng.spago.dispatching.action.util.PublishAction"
            scope="REQUEST"/>
</ACTIONS>
```

6.2 LOGIN JSP PAGE

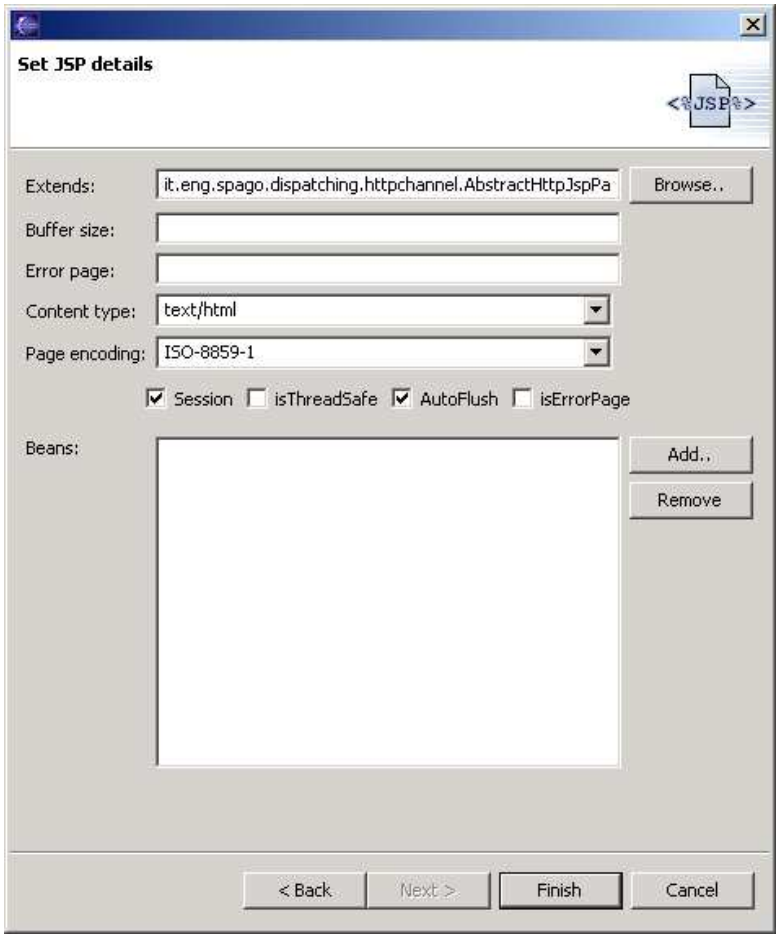
To create the Login jsp page, select *"File->New ->Lomboz JSP Wizard"*



Enter the parent folder "*SpagoDemo\SpagoDemoWeb\jsp*" and call the page *login.jsp*.



This page extends "*it.eng.spago.dispatching.httpchannel.**AbstractHttpJspPage***".



Due to a know bug of development tools, after your selection of the ***AbstractHttpJspPage*** as class to extend from, an "invalid type" error will be displayed. As work around, before pressing the "*Finish*" button you have to write some characters in the "*Buffer size*" field and then delete them.

The jsp page structure should look like this:

```
<%@ page language="java"
  extends="it.eng.spago.dispatching.httpchannel.AbstractHttpJspPage" %>
<%@ taglib uri="spagotags" prefix="spago"%>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
      <link rel="stylesheet" type="text/css"
href="../css/spago/listdetail.css"/>
      <title>Login JSP</title>
</head>
<body bgcolor="#FFFFFF" onload="checkError();">
<spago:error/>

<FORM name="login" method=post
      action="../servlet/AdapterHTTP?ACTION_NAME=LOGIN">
<table>
      <tr><th class="LISTA" height="25" colspan="2"> Login</th></tr>
      <tr><td colspan="2" class="piccolo"> </td></tr>
      <tr>
            <td class="DETAIL">UserId</td>
            <td class="DETAIL">
              <input type="text" name="user" value="" maxlength="20" size="10">
            </TD>
      </tr>
      <tr>
            <td class="DETAIL">Password</td>
            <TD class="DETAIL">
              <input type="password" name="password" value="" maxlength="20"
                        size="10">
            </TD>
      </tr>
      <tr><td>
      </table>
      <table bgcolor="#ffffff">
      <tr><td colspan="2"><input type="submit" value="Invia"></td></tr>
      </table>
      </td></tr>
</table>
</form>

</body>
</html>
```

A few notes:

- Page extends "*it.eng.spago.dispatching.httpchannel.**AbstractHttpJspPage***": in this way it is possible to access the response container, to the response, etc.
- The statement *<%@ taglib uri="spagotags" prefix="spago"%>* allows you to use the Spago tag library.
- The form's submit call the login action, as you can see from the form attribute *action="../servlet/AdapterHTTP?ACTION_NAME=LOGIN"*.

6.3 Login action

You now have to create the login action *it.eng.spago.demo.action.**LoginAction*** as follows:

```
package it.eng.spago.demo.action;

import it.eng.spago.base.SourceBean;
import it.eng.spago.configuration.ConfigSingleton;
import it.eng.spago.dispatching.action.AbstractAction;
import it.eng.spago.error.EMFErrorSeverity;
import it.eng.spago.error.EMFUserError;

public class LoginAction extends AbstractAction {

	public void service(SourceBean request, SourceBean response)
		throws Exception {
		// Reperisco i parametri presenti nel form
		String userName = (String)request.getAttribute("user");
		String password = (String)request.getAttribute("password");

		// Verifico se nel file di configurazione c'è il
		// corrispondente utente
		ConfigSingleton config = ConfigSingleton.getInstance();
		SourceBean userConfig = (SourceBean)
config.getFilteredSourceBeanAttribute("USERS.USER", "NAME", userName);

		if (userConfig == null)
			getErrorHandler().addError(new
			  EMFUserError(EMFErrorSeverity.ERROR, ERROR_USER_NOT_FOUND));
		else if ((password != null) &&
		    !password.equals((String)userConfig.getAttribute("password")))
			getErrorHandler().addError(new
			  EMFUserError(EMFErrorSeverity.ERROR, ERROR_PASSWORD_WRONG));
		else {
			response.setAttribute("name",
				(String)userConfig.getAttribute("name"));
			response.setAttribute("surname",
				(String)userConfig.getAttribute("surname"));
		}

		response.setAttribute("PUBLISHER",
			(getErrorHandler().getErrors().size() != 0) ? LOGIN_PUBLISHER
                                                    : WELCOME_PUBLISHER);
	}

	private final static int ERROR_USER_NOT_FOUND = 5000;
	private final static int ERROR_PASSWORD_WRONG = 5001;

	private final static String LOGIN_PUBLISHER = "Login";
	private final static String WELCOME_PUBLISHER = "Welcome";
}
```

The action retrieves the "*user*" and "*password*" parameters from the request and verifies if the specified user and password pair exists in the users configuration file and these matches together. Otherwise, it adds an error in the error handler (*ErrorHandler*), so the user will see the error message.

6.4 USERS CONFIGURATION FILES

This sample uses a new configuration file named "***users.xml***"; you must create it in the "*SpagoDemo\SpagoDemoWeb\conf\spago*" folderas follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<USERS>
    <USER name="Mario" surname="Rossi" password="prova1"/>
    <USER name="Giovanni" surname="Bianchi" password="prova2"/>
</USERS>
```

In "*SpagoDemo\SpagoDemoWeb\conf**master.xml***" you must add the reference to the new file through the underlain bold line, so that the Spago configurator can read it:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MASTER>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/data_access.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/actions.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/business_map.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/common.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/dispatchers.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/initializers.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/modules.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/pages.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/presentation.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/publishers.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/security.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/statements.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/tracing.xml"/>
    <CONFIGURATOR
        path="/WEB-INF/conf/spago/users.xml"/>
</MASTER>
```

6.5 MULTI-LANGUAGE CONFIGURATION

The properties files wich contains all user messages are in the "*\WEB-INF**classes**"* directory of project's template.

When an error occurs, the user messages descriptions is read by default from the properties file ***messages_en_US.properties*** located in the "*SpagoDemo\SpagoDemoWeb\WEB-INF**classes***" folder.

You must insert these files in the new project "*SpagoDemo**src**" source folder.

See the *Spago User Guide* for a different language.

In this sample we have two application's errors:

```
5000=Utente inesistente
5001=Password errata
```

6.6 PRESENTATION.XML FILE

The "*SpagoDemo\SpagoDemoWeb\WEB-INF\conf\spago**presentation.xml**" file contains the association between the actions and the corresponding publishers. Here is its configurations for this sample:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PRESENTATION>
    <MAPPING business_name="LOGIN" business_type="ACTION"
             publisher_name="GenericPublisher"/>
    <MAPPING business_name="PUBLISH" business_type="ACTION"
             publisher_name="GenericPublisher"/>
</PRESENTATION>
```

6.7 PUBLISHERS.XML FILE

The "*SpagoDemo\SpagoDemoWeb\WEB-INF\conf\spago**publishers.xml**" file contains the publishers's declarations for the different channels. In this sample, the publisher's declarations correspond only to the HTTP channel.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<PUBLISHERS>
    <PUBLISHER name="GenericPublisher">
    <RENDERING channel="HTTP" type="JAVA" mode="FORWARD">
          <RESOURCES>
          <ITEM prog="0"
resource="it.eng.spago.presentation.DefaultPublisherDispatcher">
          <CONFIG>
          <CHECKS>
          <CHECK target="Login">
                <CONDITIONS>
                <PARAMETER name="PUBLISHER" scope="SERVICE_REQUEST"
value="LOGIN" />
                </CONDITIONS>
          </CHECK>
          <CHECK target="Login">
                <CONDITIONS>
                <PARAMETER name="PUBLISHER" scope="SERVICE_RESPONSE"
value="LOGIN" />
                </CONDITIONS>
          </CHECK>
          <CHECK target="Welcome">
```

```
                    <CONDITIONS>
                    <PARAMETER name="PUBLISHER" scope="SERVICE_RESPONSE"
value="WELCOME" />
                    </CONDITIONS>
                </CHECK>
                </CHECKS>
                </CONFIG>
                </ITEM>
                </RESOURCES>
                </RENDERING>
        </PUBLISHER>
```

```
        <PUBLISHER name="Login">
            <RENDERING channel="HTTP" type="JSP" mode="FORWARD">
                <RESOURCES>
                    <ITEM prog="0" resource="/jsp/login.jsp"/>
                </RESOURCES>
            </RENDERING>
        </PUBLISHER>
```

```
        <PUBLISHER name="Welcome">
            <RENDERING channel="HTTP" type="JSP" mode="FORWARD">
                <RESOURCES>
                    <ITEM prog="0" resource="/jsp/welcome.jsp"/>
                </RESOURCES>
            </RENDERING>
        </PUBLISHER>
    </PUBLISHERS>
```

Both actions use a java-type publisher; so you can create dynamic presentation pages.

You must configure the two jsp pages.

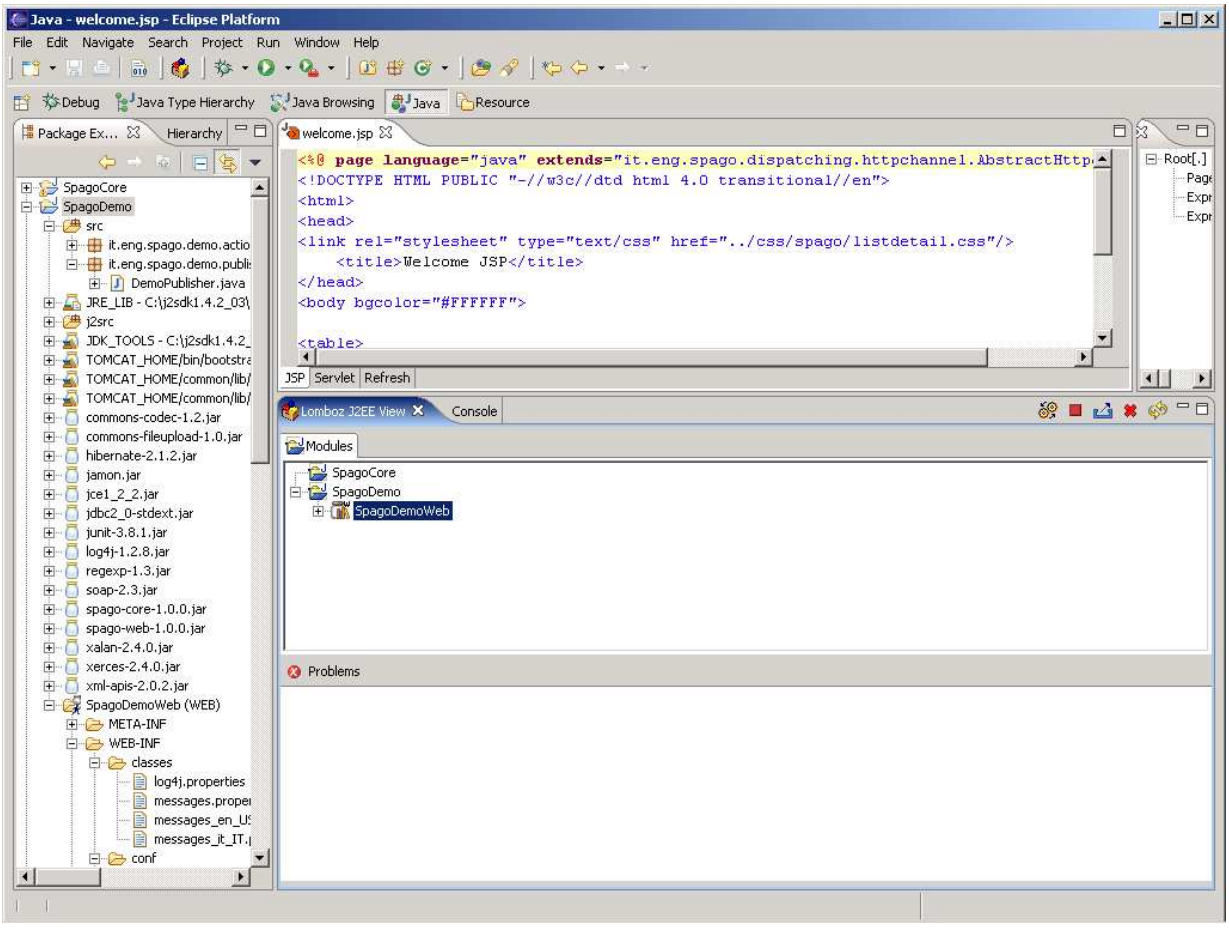
6.8 Welcome JSP page

At the end, you have to create the welcome jsp page named "*welcome.jsp*" using Lomboz:

```
<%@ page language="java"
extends="it.eng.spago.dispatching.httpchannel.AbstractHttpJspPage" %>

<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
      <link rel="stylesheet" type="text/css"
href="../css/spago/listdetail.css"/>
      <title>Welcome JSP</title>
</head>
<body bgcolor="#FFFFFF">

<table>
      <tr><th class="LISTA" height="25" colspan="2"> Welcome
      <%= getServiceResponse(request).getAttribute("name") %>
      <%= getServiceResponse(request).getAttribute("surname") %> !!</th></tr>
</table>

</body>
</html>
```

7 Deploy and Test

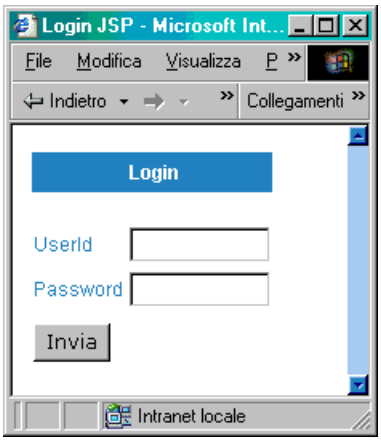
Now you have to deploy the application in the Tomcat server: from "*Java Perspective*", select  (J2EE Project Outliner); a "*Lomboz J2EE View*" panel will be displayed. Now, you can deploy application in the Tomcat server and start it.



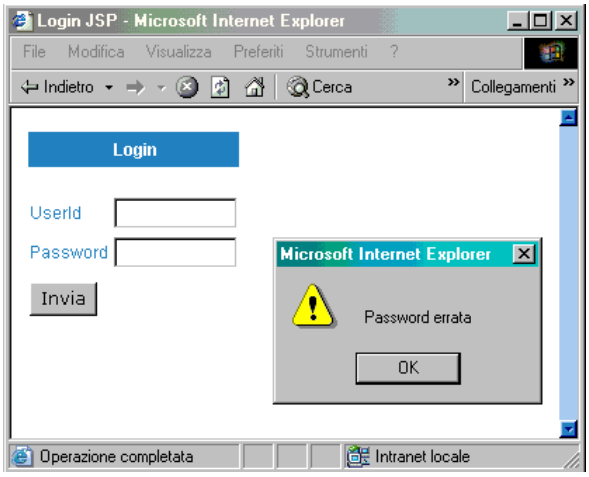
Now, with the Tomcat server working, open a browser and write the following URL:

```
http://localhost:8080/SpagoDemoWeb/servlet/AdapterHTTP?ACTION_NAME=PUBLISH
&PUBLISHER=LOGIN&NEW_SESSION=TRUE
```

The login page will be dispayed:



When an error occurs, a popup, using a Javascript, will dispay the error message retrieved from the multi-language catalogue. For example:



If everything is alright, a welcome page will be displayed showing name and surname of the logged user.