# Using Ajax

**Asynchronous JavaScript and XML** (**AJAX** or **Ajax**) is a technique for transferring data between a browser and a server asynchronously. "Asynchronously" implies that the web page continues to be processed while the request is sent from the browser to the server and a response is received. Ajax is based on JavaScript, **Document Object Model** (**DOM**), and XMLHttpRequest. Ajax provides dynamic interaction between

a browser and a server and can be used in several types of applications, such as for validating a form that requires a unique identifier without submitting the form, autocompleting input fields based on partial input, and refreshing information on a web page periodically without having to reload the web page, thus incurring less bandwidth usage.

In this chapter, we will create an Ajax application in the Eclipse IDE, compile and package the application using **Maven**, and run the web application on WildFly 8.1 with MySQL database. In this chapter, we will cover the following topics:

* Setting up the environment
* Creating a Java EE web project
* Creating a user interface
* Creating a servlet
* Deploying the Ajax application with Maven
* Running the Ajax application

## Setting up the environment

##### We need to install the following software:

* **WildFly 8.1.0.Final**: Download wildfly-8.1.0.Final.zip from

[http://wildfly.org/downloads/.](http://wildfly.org/downloads/)

**[ 123 ]**

*Using Ajax*

* **MySQL 5.6 Database-Community Edition**: Download this edition from [http://dev.mysql.com/downloads/mysql/.](http://dev.mysql.com/downloads/mysql/) When installing MySQL, also install **Connector/J**.
* **Eclipse IDE for Java EE Developers**: Download Eclipse Luna from https:// [www.eclipse.org/downloads/packages/release/Luna/SR1.](http://www.eclipse.org/downloads/packages/release/Luna/SR1)
* **JBoss Tools (Luna) 4.2.0.Final (or the latest version)**: Install this as a plugin to Eclipse from Eclipse Marketplace (<http://tools.jboss.org/downloads/> installation.html).
* **Apache Maven**: Download version 3.05 or higher from [http://maven.](http://maven/) apache.org/download.cgi.
* **Java 7**: Download Java 7 from <http://www.oracle.com/technetwork/> java/javase/downloads/index.html?ssSourceSiteId=ocomcn.

Set the JAVA\_HOME, JBOSS\_HOME, MAVEN\_HOME, and MYSQL\_HOME environment variables. Add %JAVA\_HOME%/bin, %MAVEN\_HOME%/bin, %JBOSS\_HOME%/bin, and

%MYSQL\_HOME%/bin to the PATH environment variable.

Create a WildFly 8.1.0 runtime as discussed in *Chapter 1, Getting Started with EJB 3.x*.

Create a MySQL database CATALOG with the following SQL script:CREATE TABLE Catalog(CatalogId VARCHAR(255), Journal VARCHAR(255), Publisher Varchar(255), Edition VARCHAR(255), Title Varchar(255), Author Varchar(255));

INSERT INTO Catalog VALUES('catalog1', 'Oracle Magazine', 'Oracle Publishing', 'September-October 2010', 'Using Oracle Essbase Release

11.1.2 Aggregate Storage Option Databases', 'Mark Rittman and VenkatakrishnanJanakiraman');

INSERT INTO Catalog VALUES('catalog2', 'Oracle Magazine', 'Oracle Publishing', 'July-August 2010', 'Infrastructure Software and Virtualization', 'David Baum');

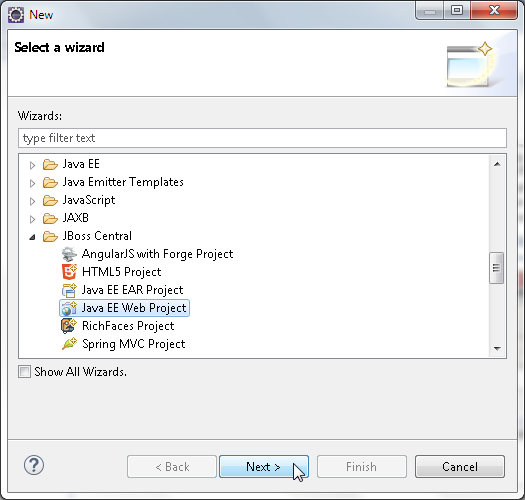
##### We will use the same MySQL data source we used in earlier chapters. The procedure to create a MySQL module, define a MySQL driver, and configure a data source is discussed in *Chapter 1, Getting Started with EJB 3.x*.

**[ 124 ]**

## Creating a Java EE web project

*Chapter 4*

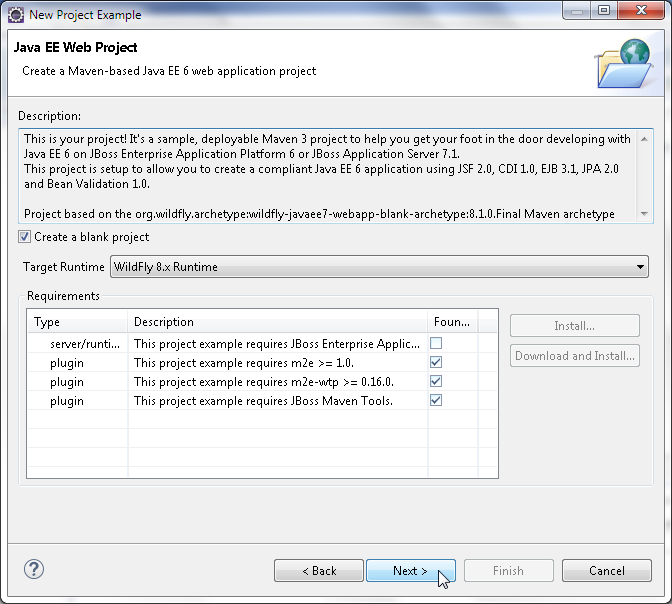
In this section, we will create a Java EE web project in Eclipse. Select **File** | **New** | **Other**. In **New**, select **JBoss Central** | **Java EE Web Project** and click on **Next**, as shown in the following screenshot:



**[ 125 ]**

*Using Ajax*

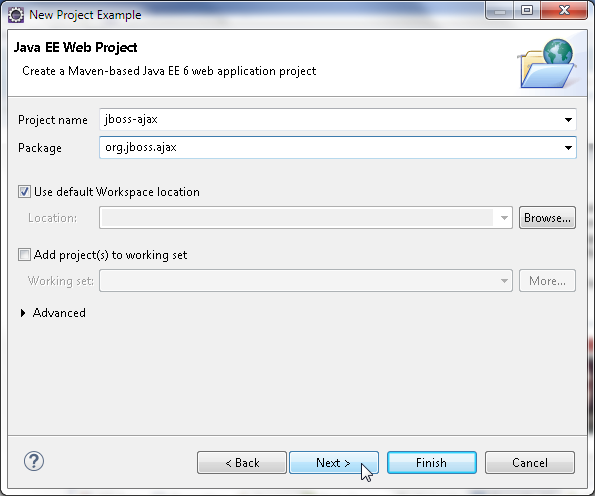
The **Java EE Web Project** wizard gets started. A test gets run for the requirements, which include the **m2e**, **m2eclipse-wtp**, and **JBoss Maven Tools** plugins. Select the **Create a blank project** checkbox and **Target Runtime** as WildFly 8.x Runtime, as shown in the following screenshot. Then click on **Next**.



**[ 126 ]**

*Chapter 4*

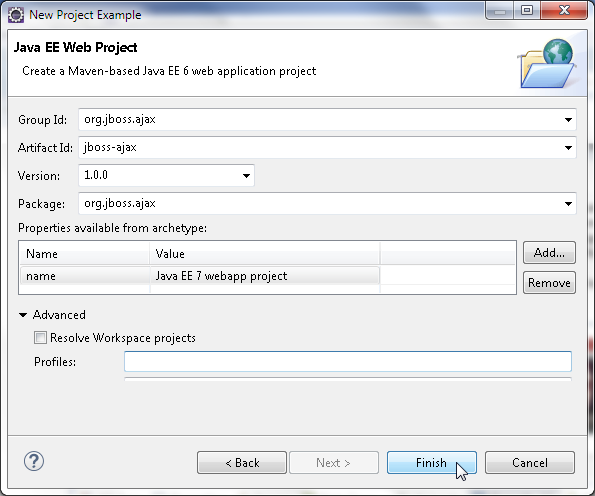
Specify **Project name** (jboss-ajax), **Package** (org.jboss.ajax), and click on **Next**, as shown in the following screenshot:



**[ 127 ]**

*Using Ajax*

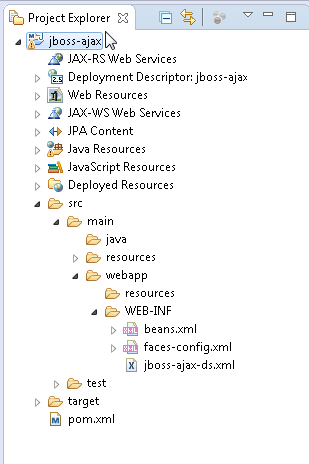
Specify **Group Id** (org.jboss.ajax), **Artifact Id** (jboss-ajax), **Version** (1.0.0), and **Package** (org.jboss.ajax), as shown in the following screenshot. After this, click on **Finish**.



**[ 128 ]**

*Chapter 4*

The jboss-ajax Java EE web project gets created, as shown in **Project Explorer** in the following screenshot. Delete the //jboss-ajax/src/main/resources/META- INF/persistence.xml configuration file as it is not used in the Ajax application:



## Creating a user interface

##### An Ajax request is initiated in a browser from a web page. In this section, we will create the user interface for the Ajax application. To initiate an Ajax request,

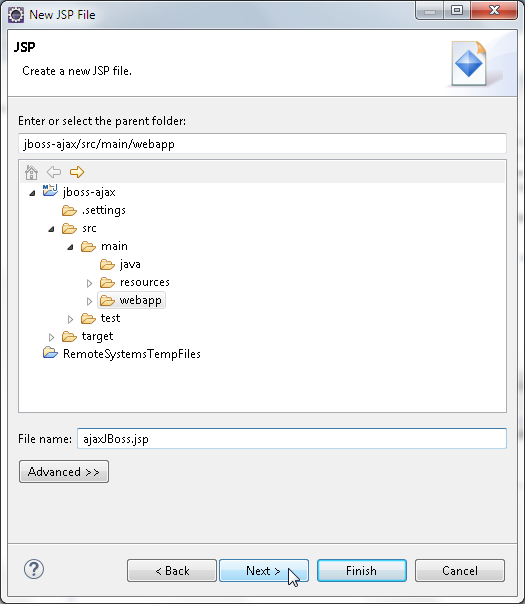
JavaScript is required, for which we will create a JSP page. We have used JSP, but another user interface technology, such as JSF, can be used instead. Select **File** | **New**

| **Other**, and in **New**, select **Web** | **JSP File** and click on **Next**.

**[ 129 ]**

*Using Ajax*

In the **New JSP File** wizard, select the webapp folder, specify **File name** (ajaxJBoss. jsp), and click on **Next,** as shown in the following screenshot:



In **Select JSP Template**, select the **New JSP File (html)** template, and click on **Finish**.

##### The ajaxJBoss.jsp file gets added to the webapp folder. In ajaxJBoss.jsp, add an HTML form to create a catalog entry. The input form consists of input fields for Catalog ID, journal, publisher, edition, title, and author. The **Catalog ID** field requires a unique field value. In a form without Ajax, we would specify a Catalog

ID value and the other field values and submit the form with a **Submit** button. If the Catalog ID is unique, a new catalog entry would get created, but if the Catalog ID already exists in the database, an error message would get displayed and the form would be required to be refilled and resubmitted. With Ajax, the Catalog ID value can be validated as the value is specified, thus preempting the need to resubmit the form. In the Ajax application, the input Catalog ID value is sent to the server as the value is specified using an Ajax request and an HTTP servlet immediately returns an XML message about the validity of the input data.

**[ 130 ]**

*Chapter 4*

##### To send XMLHttpRequest and receive a response, we will use the following procedure:

1. Invoke a JavaScript function from an HTML event, such as onkeyup.
2. Create an XMLHttpRequest object in the JavaScript function.
3. Open an XMLHttpRequest request.

##### Register a callback event handler, which gets invoked when the request is complete, with the XMLHttpRequest object.

1. Send an XMLHttpRequest request to the server asynchronously.
2. Process the request on the server; for the server, the asynchronous request is

just like any other HTTP request.

1. Send an XML message response back to the browser.
2. Receive the XML response and display a message on the web page without

reloading the web page.

In ajaxJBoss.jsp, add an HTML form with input fields for a catalog entry. Set the action for the form as AjaxFormServlet, which is mapped to invoke a servlet, as discussed later. Add <table/> within <form/> and add a <input/> field for

##### a Catalog ID. Add a **Submit** button to submit the form to the server using the HTTP POST method. In the <input/> field, set the onkeyup event handler to a validateCatalogId() JavaScript function, which we will add to ajaxJBoss. jsp. Include <div/> in the table row for the Catalog ID input field; <div/> will be used to display a message about the validity of the Catalog ID. Here's the code that encapsulates the discussion in this paragraph:

<table>

<tr>

<td>Catalog Id:</td><td><input type="text" size="20" id="catalogId" name="catalogId" onkeyup="validateCatalogId()">

</td>

<td>

<div id="validationMessage"></div>

</td>

</tr>

…

</table>

##### Similarly, add the other input fields for a catalog. Each time the onkeyup event is generated, the validateCatalogId function gets invoked.

**[ 131 ]**

*Using Ajax*

Next, create a validateCatalogId JavaScript function. In the validateCatalogId() JavaScript function, create a new XMLHttpRequest object. If a browser supports the XMLHttpRequest object as an ActiveX object (as in IE 6), the procedure to create

an XMLHttpRequest object is different than the procedure if the XMLHttpRequest object is a native object; a window object property (as in IE 7 and later, and other browsers). Create an init() function within the validateCatalogId function and create an XMLHttpRequest object for both types of browsers (those supporting / not supporting XMLHttpRequest as a native object). The following is the code for the discussion in this paragraph:

function validateCatalogId(){ var xmlHttpRequest=init(); function init(){

if (window.XMLHttpRequest) { return new XMLHttpRequest();

} else if (window.ActiveXObject) {

return new ActiveXObject("Microsoft.XMLHTTP");

}

}

}

Next, open XMLHttpRequest using the open() method, url [, async = true [, user = null [, password = null]]]). Set the HTTP method to GET for the browser to receive a response from the server. The server URL to which XMLHttpRequest

is to be sent consists of a servlet mapping to invoke a servlet to process the request and the CatalogId request parameter. In the example application, we will invoke AjaxFormServlet, which is mapped to /AjaxFormServlet in web.xml. Encode the request parameter CatalogId using the encodeURIComponent(string) method, which encodes the CatalogId value to UTF-8 (https://developer.mozilla.org/ en-US/docs/JavaScript/Reference/Global\_Objects/encodeURIComponent). By default, the user and password are set an empty string. The following code shows this action performed on the example application:

varcatalogId=document.getElementById("catalogId"); xmlHttpRequest.open("GET", "AjaxFormServlet?catalogId="+ encodeURIComponent(catalogId.value), true);

##### We need to know when a request has been completed so that we can process

the response. Register a callback event handler, processRequest, with the XMLHttpRequest object using the onreadystatechange property. The JavaScript callback function processRequest, which we will add later, gets invoked whenever the value of the readyState property changes, as shown in the following line

##### of code:

xmlHttpRequest.onreadystatechange=processRequest;

**[ 132 ]**

*Chapter 4*

##### Send an Ajax request using the send() method (as the HTTP method is GET, data sent with the send method is set to null), as shown in the following line of code:

xmlHttpRequest.send(null);

With an asynchronous request, the send() method returns immediately. The processRequest function gets invoked each time the value of the readyState property changes. In the processRequest function, retrieve the readyState property value. When the request has loaded completely corresponding to the readyState property value 4 and the HTTP status OK, invoke the processResponse JavaScript function to process the response from the server:

function processRequest(){ if(xmlHttpRequest.readyState==4){

if(xmlHttpRequest.status==200){ processResponse();

}

}

}

The XMLHttpRequest request invokes AjaxFormServlet. The servlet processes the

##### request and returns a response as an XML message of the following format:

<catalog>

<valid></valid>

<journal></journal>

<publisher></publisher>

<edition></edition>

<title></title>

<author></author>

</catalog>

We will discuss the server-side processing of the request in the *Creating a Servlet* section. The XMLHttpRequest request's response from the server is processed, and if the instructions indicate that the CatalogId input is valid, a message **Catalog Id is Valid** is displayed. XMLHttpRequest will be sent to the server and a response is received with each modification in the catalogId input field.

##### Next, add the JavaScript function to process the response processResponse(). In the processRequest() JavaScript function, when the HTTP request has loaded completely, which corresponds to the readyState property value 4 and the HTTP status OK, which in turn corresponds to the status property value 200, the processResponse() JavaScript function gets invoked. In the processResponse() function, obtain the value of the responseXML property, which contains the XML string returned from the server. This is shown in the following line of code:

Var xmlMessage=xmlHttpRequest.responseXML;

**[ 133 ]**

*Using Ajax*

The responseXML property contains the <valid/> element, which indicates the validity of the CatalogId value specified in the input form. Obtain the value of the <valid/> element using the getElementsByTagName(string) method. This is shown in the following lines of code:

var valid=xmlMessage.getElementsByTagName("valid")[0].firstChild. nodeValue;

If the <valid/> element value is true, set the validationMessage div tag to Catalog Id is Valid, and enable the submit button in the input form. Also, set the value of the form fields to an empty string so that new input values can be specified:

if(valid=="true"){ varvalidationMessage=document.getElementById

("validationMessage");

validationMessage.innerHTML = "Catalog Id is Valid"; document.getElementById("submitForm").disabled = false;

……

}

##### If the Catalog ID value is valid, a new catalog entry can be created by adding values for the different values of a catalog entry. Submit the form with the **Submit** button. If the <valid/> element value is false, set the validationMessage div tag in the CatalogID field row to Catalog Id is not Valid, and disable the **Submit** button. Set the values of other input fields to the values returned in the XML message from the server. For example, the journal field value is set as follows, setting the values of the other fields as an example of autocompletion with Ajax:

if(valid=="false"){

var validationMessage=document.getElementById("validationMessage"); validationMessage.innerHTML = "Catalog Id is not Valid"; document.getElementById("submitForm").disabled = true;

var journal=xmlMessage.getElementsByTagName ("journal")[0].firstChild.nodeValue;

…

var journalElement=document.getElementById("journal"); journalElement.value = journal;

}

**[ 134 ]**

*Chapter 4*

The ajaxJBoss.jsp file is listed as follows:

<html>

<head>

<script type="text/javascript"> function validateCatalogId(){

var xmlHttpRequest=init(); function init(){

if (window.XMLHttpRequest) { return new XMLHttpRequest();

} else if (window.ActiveXObject) {

return new ActiveXObject("Microsoft.XMLHTTP");

}

}

var catalogId=document.getElementById("catalogId"); xmlHttpRequest.open("GET", "AjaxFormServlet?catalogId="+

encodeURIComponent(catalogId.value), true);

xmlHttpRequest.onreadystatechange=processRequest; xmlHttpRequest.send(null);

function processRequest(){ if(xmlHttpRequest.readyState==4){ if(xmlHttpRequest.status==200){ processResponse();

}

}

}

function processResponse(){

var xmlMessage=xmlHttpRequest.responseXML;

var valid=xmlMessage.getElementsByTagName("valid")[0]. firstChild.nodeValue;

if(valid=="true"){

var validationMessage=document.getElementById("validationMessa

ge");

validationMessage.innerHTML = "Catalog Id is Valid"; document.getElementById("submitForm").disabled = false; var journalElement=document.getElementById("journal"); journalElement.value = "";

var publisherElement=document.getElementById("publisher"); publisherElement.value = "";

var editionElement=document.getElementById("edition"); editionElement.value = "";

var titleElement=document.getElementById("title");

**[ 135 ]**

*Using Ajax*

titleElement.value = "";

var authorElement=document.getElementById("author"); authorElement.value = "";

}

if(valid=="false"){

var validationMessage=document.getElementById("validationMessa

ge");

validationMessage.innerHTML = "Catalog Id is not Valid"; document.getElementById("submitForm").disabled = true;

var journal=xmlMessage.getElementsByTagName("journal")[0]. firstChild.nodeValue;

var publisher=xmlMessage.getElementsByTagName("publisher")[0]. firstChild.nodeValue;

var edition=xmlMessage.getElementsByTagName("edition")[0]. firstChild.nodeValue;

var title=xmlMessage.getElementsByTagName("title")[0]. firstChild.nodeValue;

var author=xmlMessage.getElementsByTagName("author")[0]. firstChild.nodeValue;

var journalElement=document.getElementById("journal"); journalElement.value = journal;

var publisherElement=document.getElementById("publisher"); publisherElement.value = publisher;

var editionElement=document.getElementById("edition"); editionElement.value = edition;

var titleElement=document.getElementById("title"); titleElement.value = title;

var authorElement=document.getElementById("author"); authorElement.value = author;

}

}

}

</script>

</head>

<body>

<h1>Form for Catalog Entry</h1>

<form name="AjaxFormServlet" action="AjaxFormServlet" method="post">

**[ 136 ]**

*Chapter 4*

<table>

<tr>

<td>Catalog Id:</td>

<td><input type="text" size="20" id="catalogId" name="catalogId"

onkeyup="validateCatalogId()"></td>

<td>

<div id="validationMessage"></div>

</td>

</tr>

<tr>

<td>Journal:</td>

<td><input type="text" size="20"

id="journal" name="journal"></td>

</tr>

<tr>

<td>Publisher:</td>

<td><input type="text" size="20" id="publisher" name="publisher"></td>

</tr>

<tr>

<td>Edition:</td>

<td><input type="text" size="20"

id="edition" name="edition"></td>

</tr>

<tr>

<td>Title:</td>

<td><input type="text" size="20"

id="title" name="title"></td>

</tr>

<tr>

<td>Author:</td>

<td><input type="text" size="20"

**[ 137 ]**

*Using Ajax*

id="author" name="author"></td>

</tr>

<tr>

<td><input type="submit" value="Create Catalog" id="submitForm" name="submitForm"></td>

</tr>

</table>

</form>

</body>

</html>

##### Add a catalog.jsp JSP to output a message that a catalog entry has been created without error, as follows:

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" ["http://www.w3.org/TR/html4/loose.dtd">](http://www.w3.org/TR/html4/loose.dtd)

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Catalog entry created without error</title>

</head>

<body>

Catalog entry created without error

</body>

</html>

##### Add another error.jsp JSP to indicate an error message, as follows:

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" ["http://www.w3.org/TR/html4/loose.dtd">](http://www.w3.org/TR/html4/loose.dtd)

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Error page</title>

</head>

<body>

Error Page

</body>

</html>

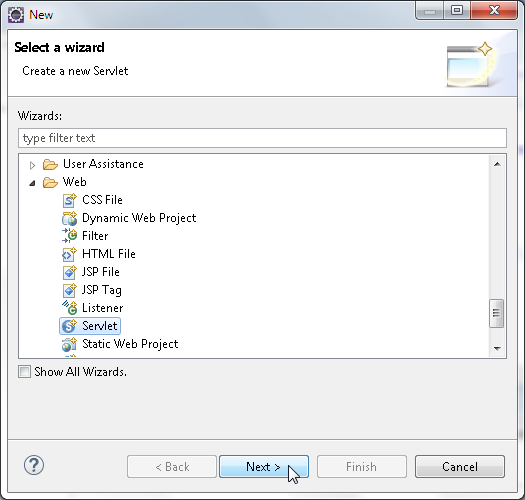
**[ 138 ]**

## Creating a servlet

*Chapter 4*

##### In this section, we will create a servlet to process an Ajax request. Select **File** | **New** |

**Other**, and in **New**, select **Web** | **Servlet**, which is shown as follows. Then, click on **Next**.

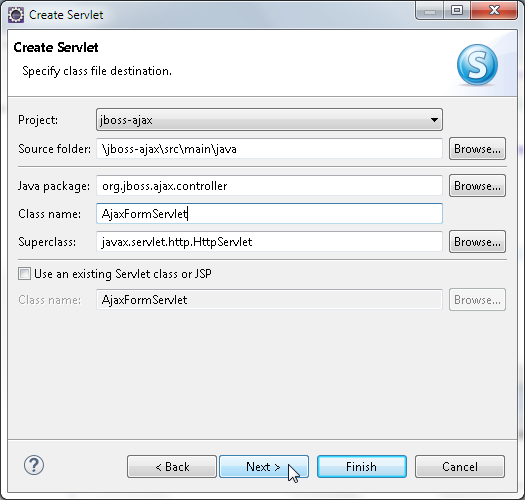


**[ 139 ]**

*Using Ajax*

The **Create Servlet** wizard gets started. Select **Project** as jboss-ajax, **Source folder**

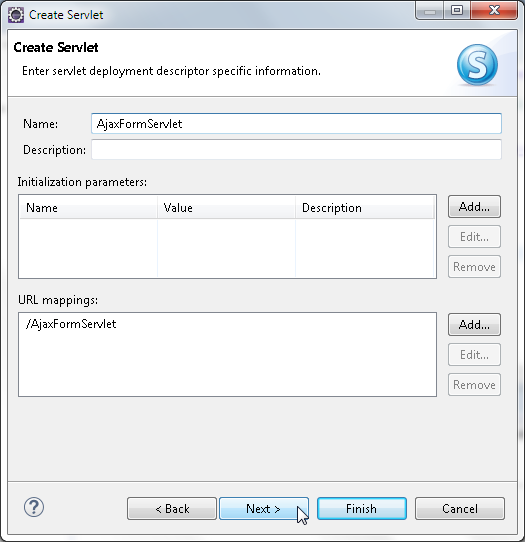
as src\main\java, **Java package** as org.jboss.ajax.controller, **Class name** as AjaxFormServlet, and **Superclass** as javax.servlet.http.HttpServlet, as shown in the following screenshot. Then click on **Next**.



**[ 140 ]**

*Chapter 4*

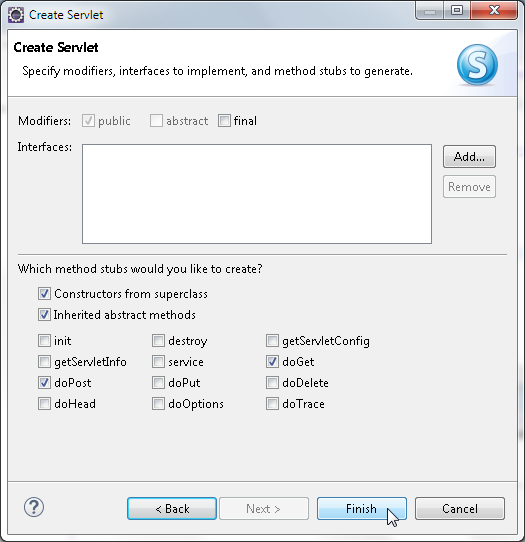
Specify URL mappings as AjaxFormServlet and click on **Next**, as shown in the following screenshot:



**[ 141 ]**

*Using Ajax*

##### Select the doGet and doPost methods to create the servlet, as shown in the following screenshot. Once this is done, click on **Finish**.



AjaxFormServlet gets created and the servlet gets configured in web.xml, including a URL mapping to /AjaxFormServlet. We included /AjaxFormServlet in the URL to send an XMLHttpRequest request to invoke AjaxFormServlet. The web.xml file is listed as follows:

<?xml version="1.0" encoding="UTF-8"?>

<web-app [xmlns="http://xmlns.jcp.org/xml/ns/javaee"](http://xmlns.jcp.org/xml/ns/javaee) xmlns:xsi="http:// [www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xsi:schemaLocation="..." version="3.1">

<display-name>jboss-ajax</display-name>

<servlet>

<description></description>

**[ 142 ]**

*Chapter 4*

<display-name>AjaxFormServlet</display-name>

<servlet-name>AjaxFormServlet</servlet-name>

<servlet-class>org.jboss.ajax.controller.AjaxFormServlet</servlet- class>

</servlet>

<servlet-mapping>

<servlet-name>AjaxFormServlet</servlet-name>

<url-pattern>/AjaxFormServlet</url-pattern>

</servlet-mapping>

</web-app>

##### As the HTTP method is GET, the doGet() method of the servlet gets invoked. In the doGet method, retrieve the value of the catalogId parameter, as shown in the following line of code:

String catalogId = request.getParameter("catalogId");

##### Apply the business logic on the catalogId value to validate the value. We have used the business logic that the value must be unique to be valid, which implies the same value must not already be in the database. Create a DataSource object using a JNDI lookup with an InitialContext object on the java:jboss/datasources/ MySQLDS data source.

Create a Connection object from the DataSource object using the getConnection() method. Using the CatalogId value specified in the input form, create a SQL query to retrieve the data from the database. Create a PreparedStatement object from

the Connection object using the prepareStatement(String) method. Run the SQL query using the executeQuery() method to obtain a ResultSet object. If the ResultSet object is empty, it implies that the CatalogId field value is not defined in the Catalog database table; the CatalogId field value is valid. If the ResultSet object contains data, it implies that the CatalogId value already exists in the database; the CatalogId field value is not valid.

##### Next, construct an XML string to return to the server. If CatalogId is not valid, construct an XML string that includes the different field values for the catalog entry as XML elements. The XML string is required to have a root element, catalog, for example. Include a <valid> </valid> element that specifies the validity of the CatalogId field value with a boolean value. If the CatalogId value is valid, add only the <valid> </valid> element to the XML string, as shown in the following code snippet (the variable rs represents ResultSet):

if (rs.next()) {

out.println("<catalog>" + "<valid>false</valid>" + "<journal>" + rs.getString(2) + "</journal>" + "<publisher>" +

rs.getString(3) + "</publisher>" + "<edition>" + rs.getString(4) + "</edition>" + "<title>" +

**[ 143 ]**

*Using Ajax*

rs.getString(5) + "</title>" + "<author>" + rs.getString(6) + "</author>" + "</catalog>");

} else { out.println("<valid>true</valid>");

}

Set the content type of HttpServletResponse to text/xml because the response

##### to an Ajax request is in the XML format, and set the Cache-Control header to

no-cache to prevent JSPs and servlets from being cached. As the Ajax response is updated with each request, caching must be disabled to prevent a cached response from being reserved, as follows:

response.setContentType("text/xml"); response.setHeader("Cache-Control", "no-cache");

##### If the CatalogId field value does not exist in the database, the input form with field values for a new catalog entry can be submitted using the POST method. In the doPost method in the servlet, create a JDBC connection to the MySQL database as in the doGet method, and add a catalog entry with an INSERT SQL statement.

The FormServlet.java Ajax is listed as follows:

package org.jboss.ajax.controller;

import java.io.\*; import java.sql.\*;

import javax.naming.InitialContext; import javax.servlet.ServletException; import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import javax.sql.DataSource;

public class AjaxFormServlet extends HttpServlet {

##### The doGet method is invoked with an asynchronous request sent using the HTTP GET method. Run a SQL query using Catalog Id, which is specified in the input form to generate a result set. Set headers for the HttpServletResponse object, and create a PrintWriter object from the HttpServletResponse object. Construct an output as an XML response, as shown in the following code:

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException { try {

// Obtain value of Catalog Id field to be validated.

**[ 144 ]**

*Chapter 4*

String catalogId = request.getParameter("catalogId");

// Obtain Connection

InitialContext initialContext = new InitialContext();

DataSource ds = (DataSource)initialContext.lookup("java:jboss/ datasources/MySQLDS");

java.sql.Connection conn = ds.getConnection();

// Obtain result set

PreparedStatement pstmt = conn.prepareStatement("SELECT \* from CATALOG WHERE CatalogId = ?");

pstmt.setString(1, catalogId); ResultSet rs = pstmt.executeQuery();

// set headers before accessing the Writer response.setContentType("text/xml"); response.setHeader("Cache-Control", "no-cache");

PrintWriter out = response.getWriter();

// then send the response

// If result set is empty set valid element to true if (rs.next()) {

out.println("<catalog>" + "<valid>false</valid>" + "<journal>"

+ rs.getString(2) + "</journal>" + "<publisher>"

+ rs.getString(3) + "</publisher>" + "<edition>"

+ rs.getString(4) + "</edition>" + "<title>"

+ rs.getString(5) + "</title>" + "<author>"

+ rs.getString(6) + "</author>" + "</catalog>");

} else { out.println("<valid>true</valid>");

}

rs.close();

stmt.close();

conn.close();

} catch (javax.naming.NamingException e) {System.err.println(e. getMessage());

} catch (SQLException e) {System.err.println(e.getMessage());

}

}

**[ 145 ]**

*Using Ajax*

The doPost() method is used to create a new catalog entry. Create an InitialContext object. With a JNDI lookup, create a DataSource object. Obtain a Connection object from the DataSource object using the getConnection() method. Create a Statement object using the createStatement() method of the Connection class. PreparedStatement can be used instead of Statement. Create a SQL string from values retrieved from the input form. Run the SQL statement using the execute() method. If the SQL statement runs without error, redirect the response

to catalogentrycreated.jsp. If an error is generated, redirect the response to

error.jsp, as shown in the following code:

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException { try {

// Obtain Connection

InitialContext initialContext = new InitialContext();

DataSource ds = (DataSource) initialContext.lookup("java:jboss/ datasources/MySQLDS");

java.sql.Connection conn = ds.getConnection();

String catalogId = request.getParameter("catalogId"); String journal = request.getParameter("journal"); String publisher = request.getParameter("publisher"); String edition = request.getParameter("edition"); String title = request.getParameter("title");

String author = request.getParameter("author");

Statement stmt = conn.createStatement();

String sql = "INSERT INTO Catalog VALUES(" + "\'" + catalogId

+ "\'" + "," + "\'" + journal + "\'" + "," + "\'"

+ publisher + "\'" + "," + "\'" + edition + "\'" + ","

+ "\'" + title + "\'" + "," + "\'" + author + "\'" + ")"; stmt.execute(sql); response.sendRedirect("catalogentrycreated.jsp");

stmt.close();

conn.close();

} catch (javax.naming.NamingException e) { response.sendRedirect("error.jsp");

} catch (SQLException e) { response.sendRedirect("error.jsp");

**[ 146 ]**

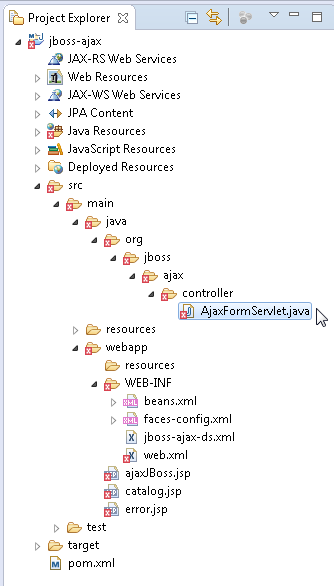
*Chapter 4*

}

}

}

The AjaxFormServlet class is shown in **Package Explorer** in the following screenshot. The errors shown in the listing will be removed in the next section once the dependencies are satisfied through Maven.



## Deploying the Ajax application

**with Maven**

##### In this section, we will compile, package, and deploy the Ajax application to WildFly 8.1 using the Maven build tool. The information about the project and the configuration details are specified in pom.xml in the root directory of the Ajax application.

**[ 147 ]**

*Using Ajax*

##### As we are using the MySQL database, add a dependency on the MySQL JDBC Java

Connector, as follows:

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.22</version>

</dependency>

##### Add the dependency for the Servlet 3.1 API, as follows:

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

##### Add the Maven compiler plugin and the Maven WAR plugin in the build element. In the configuration for the Maven WAR plugin, specify the output directory as the deployments directory of WildFly 8.1. The pom.xml file is listed as follows:

<?xml version="1.0" encoding="UTF-8"?>

<project [xmlns="http://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0) xmlns:xsi="http:// [www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) [xsi:schemaLocation="http://maven.](http://maven/) apache.org/POM/4.0.0 [http://maven.apache.org/maven-v4\_0\_0.xsd">](http://maven.apache.org/maven-v4_0_0.xsd)

<modelVersion>4.0.0</modelVersion>

<groupId>org.jboss.ajax</groupId>

<artifactId>jboss-ajax</artifactId>

<version>1.0.0</version>

<packaging>war</packaging>

<name>WildFly Ajax</name>

<description>A starter Java EE 7 webapp project for use on JBoss WildFly / WildFly, generated from the jboss-javaee6-webapp archetype</ description>

[<url>http://wildfly.org</url>](http://wildfly.org/)

<licenses>

<license>

<name>Apache License, Version 2.0</name>

<distribution>repo</distribution>

[<url>http://www.apache.org/licenses/LICENSE-2.0.html</url>](http://www.apache.org/licenses/LICENSE-2.0.html)

</license>

</licenses>

<properties>

<!-- Explicitly declaring the source encoding eliminates the following message: -->

**[ 148 ]**

*Chapter 4*

<!-- [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent! -->

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<!-- JBoss dependency versions -->

<version.wildfly.maven.plugin>1.0.2.Final</version.wildfly.maven. plugin>

<!-- Define the version of the JBoss BOMs we want to import to specify tested stacks. -->

<version.jboss.bom>8.1.0.Final</version.jboss.bom>

<version.arquillian.container>8.0.0.Final</version.arquillian. container>

<!-- other plugin versions -->

<version.compiler.plugin>3.1</version.compiler.plugin>

<version.surefire.plugin>2.16</version.surefire.plugin>

<version.war.plugin>2.1.1</version.war.plugin>

<!-- maven-compiler-plugin -->

<maven.compiler.target>1.7</maven.compiler.target>

<maven.compiler.source>1.7</maven.compiler.source>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.wildfly.bom</groupId>

<artifactId>jboss-javaee-7.0-with-tools</artifactId>

<version>${version.jboss.bom}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>org.wildfly.bom</groupId>

<artifactId>jboss-javaee-7.0-with-hibernate</artifactId>

<version>${version.jboss.bom}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- First declare the APIs we depend on and need for compilation.

All of them are provided by JBoss WildFly -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

**[ 149 ]**

*Using Ajax*

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.22</version>

</dependency>

</dependencies>

<build>

<!-- Maven will append the version to the finalName (which is the name given to the generated war, and hence the context root) -->

<finalName>${project.artifactId}</finalName>

<plugins>

<!-- Compiler plugin enforces Java 1.6 compatibility and activates annotation processors -->

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>${version.compiler.plugin}</version>

<configuration>

<source>${maven.compiler.source}</source>

<target>${maven.compiler.target}</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>${version.war.plugin}</version>

<configuration>

<outputDirectory>C:\wildfly-8.1.0.Final\standalone\ deployments</outputDirectory>

<!-- Java EE 7 doesn't require web.xml, Maven needs to catch

up! -->

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

<!-- The WildFly plugin deploys your war to a local WildFly container -->

<!-- To use, run: mvn package wildfly:deploy -->

<plugin>

<groupId>org.wildfly.plugins</groupId>

<artifactId>wildfly-maven-plugin</artifactId>

<version>${version.wildfly.maven.plugin}</version>

</plugin>

</plugins>

</build>

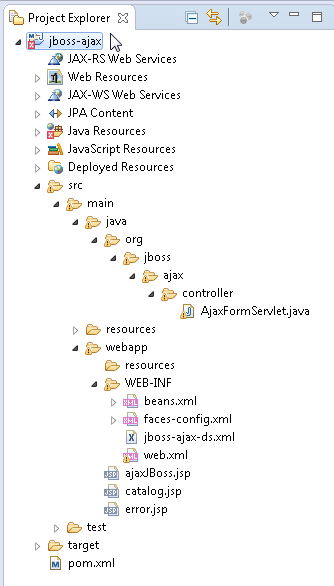
</project>

**[ 150 ]**

*Chapter 4*

##### After the dependencies have been added to pom.xml, the errors in the JSPs and the

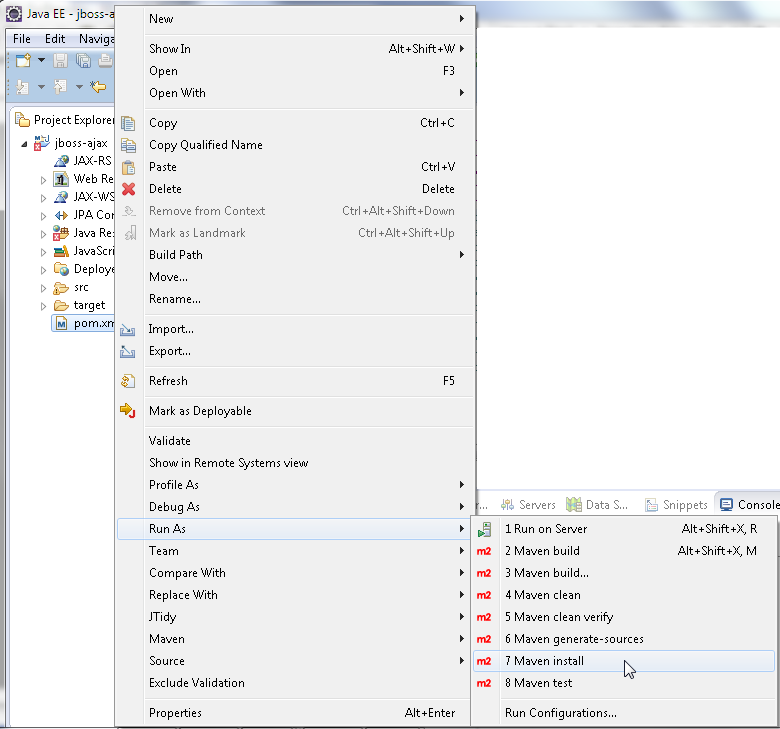
servlet get removed, as shown in the following screenshot:



**[ 151 ]**

*Using Ajax*

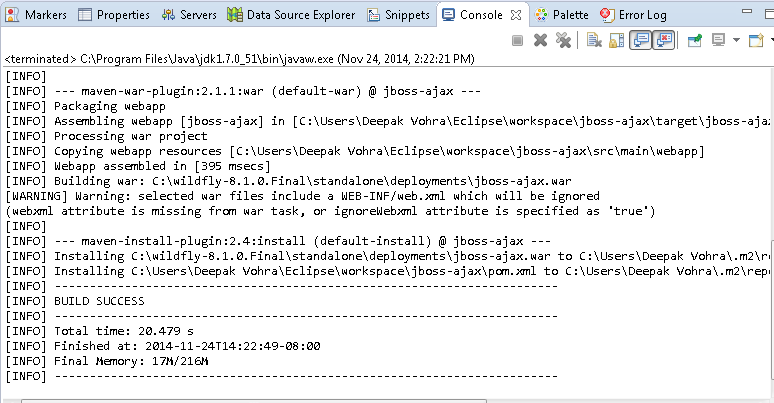
Right-click on pom.xml and select **Run As** | **Maven install**, as shown in the following screenshot:



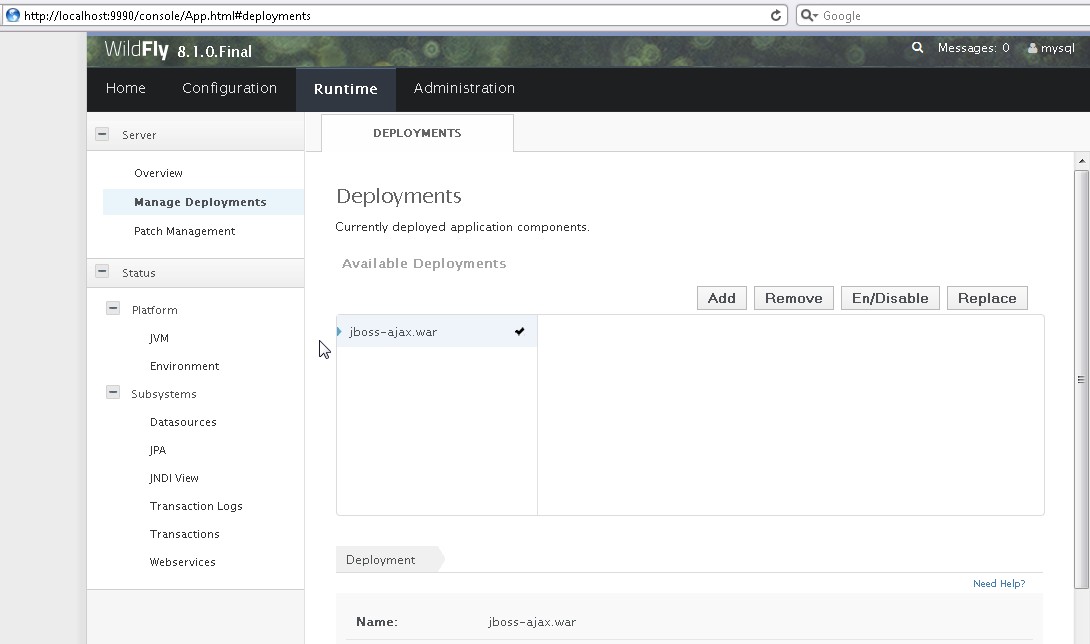
The jboss-ajax application gets compiled and packaged into jboss-ajax.war, which gets the output to the deployments directory. The Maven build outputs the message BUILD SUCCESS, as shown in the following screenshot:

**[ 152 ]**

*Chapter 4*



Start WildFly 8.1 if it is not already started. The jboss-ajax.war gets deployed to WildFly 8.1 and the web context root |jboss-ajax gets registered. The jboss-ajax. war gets deployed and gets listed in WildFly 8.1. **Administration Console**, as shown in the following screenshot:

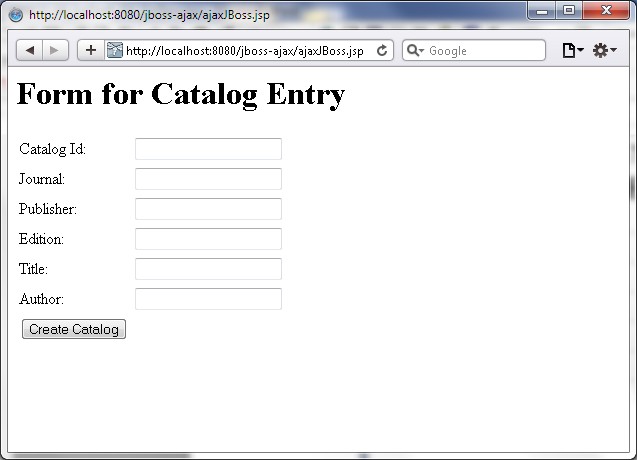


**[ 153 ]**

*Using Ajax*

## Running the Ajax application

In this section, we will run the Ajax application. Run ajaxBoss.jsp using the URL http://localhost:8080/jboss-ajax/ajaxJBoss.jsp. The input form for a catalog entry gets displayed, as shown in the following screenshot:

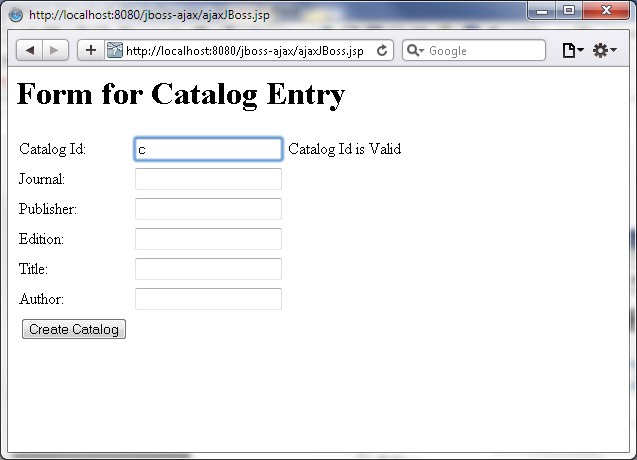


##### Start to specify a **Catalog Id** value. As an Ajax request is sent to the server, with each

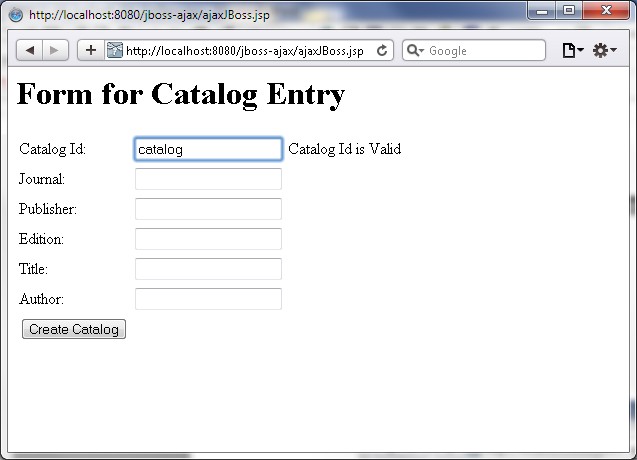
modification to the **Catalog Id** value, a response is returned from the server and a message gets displayed about the validity of **Catalog Id**, which is shown in the following screenshot:

**[ 154 ]**

*Chapter 4*



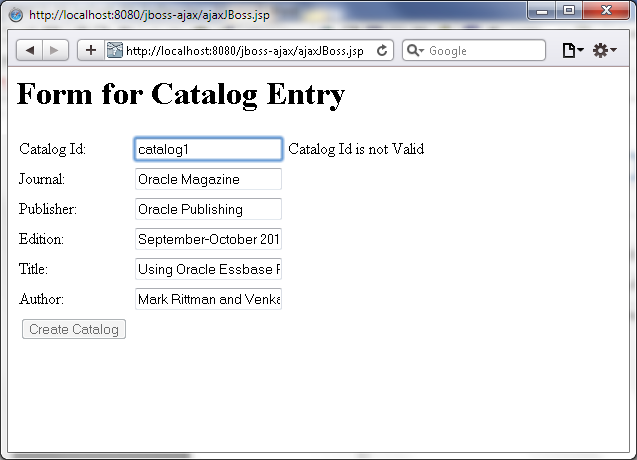
##### The business logic used for **Catalog Id** to be valid is that the value should be unique, but another logic can be used instead. **Catalog Id** (catalog) is still valid, as shown in the following screenshot:



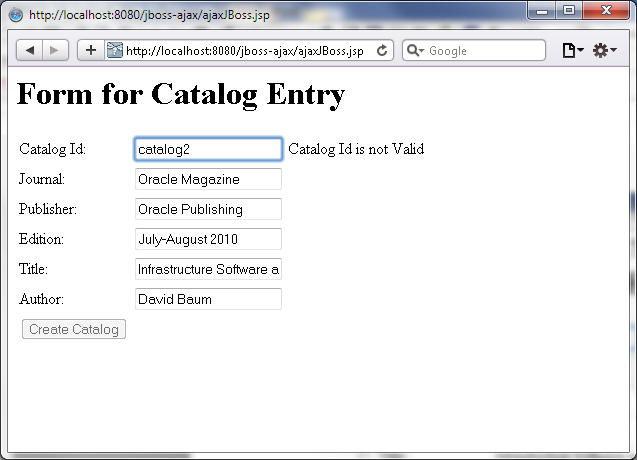
**[ 155 ]**

*Using Ajax*

##### Specify a **Catalog Id** value that is already in the MySQL database, catalog1 for example. A message **Catalog Id is not Valid** gets displayed, and the input fields get filled with the catalog entry's field values, as follows:



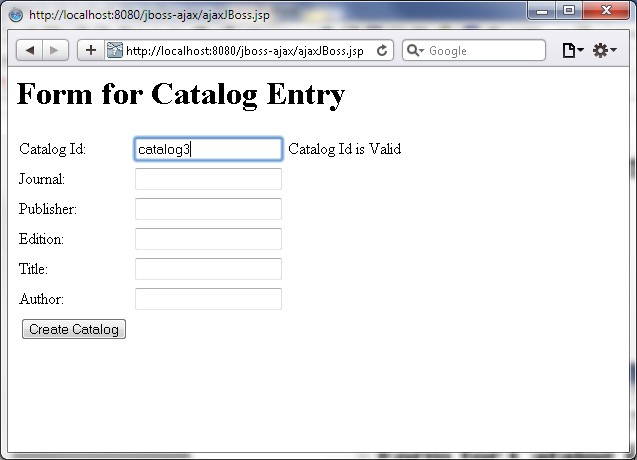
Specify catalog2 as **Catalog Id**, which is also not valid, as shown in the following screenshot:



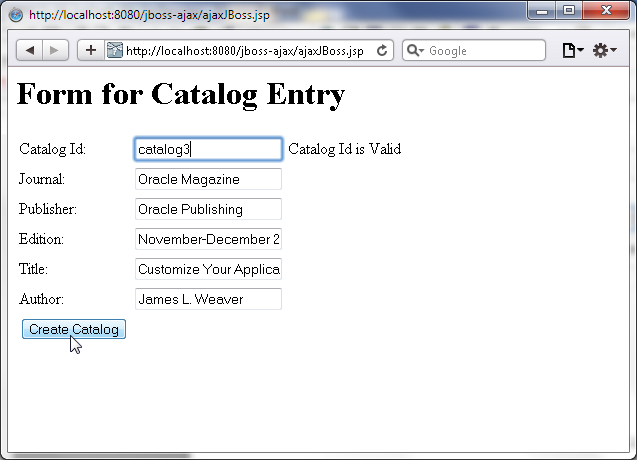
**[ 156 ]**

*Chapter 4*

Specify catalog3 as **Catalog Id**. As the catalog3 value is not already in the database, **Catalog Id is valid**, as shown in the following screenshot:



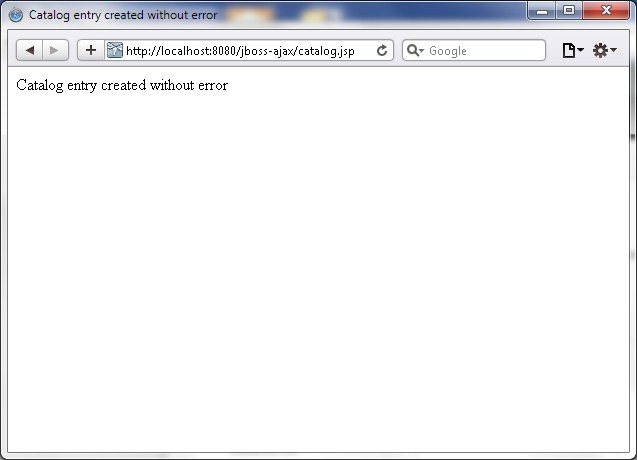
##### Add field values for a new catalog entry and click on **Submit**, as shown in the following screenshot:



**[ 157 ]**

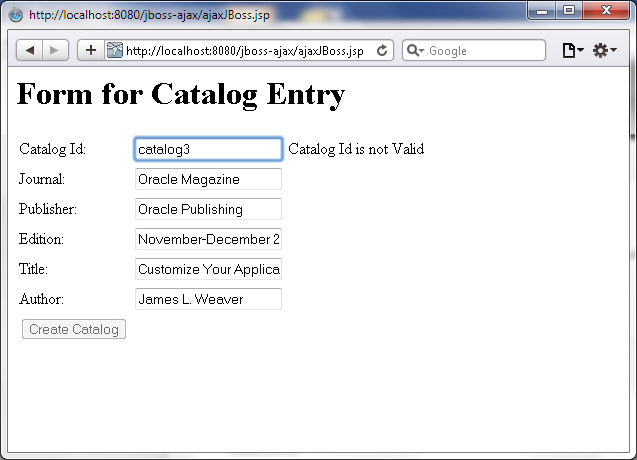
*Using Ajax*

##### A new catalog entry gets created as indicated by the message shown in the following screenshot:



If the catalog3 value is re-added to the **Catalog Id** field, the **Catalog Id is not Valid**

##### message gets displayed, as shown in the following screenshot:



**[ 158 ]**

*Chapter 4*

## Summary

##### In this chapter, we learned how to develop an Ajax application in Eclipse. We

compiled and packaged the application with the Maven build tool and deployed

the application to WildFly 8.1. We ran the Ajax application in a browser with the MySQL database. We discussed only some of the methods and attributes of the XMLHttpRequest object. For complete information on XMLHttpRequest, refer to <http://www.w3.org/TR/XMLHttpRequest/>and https://developer.

mozilla.org/en-US/docs/DOM/XMLHttpRequest?redirectlocale=en- US&redirectslug=XMLHttpRequest.

##### In the next chapter, we will discuss the GWT web framework, which provides Ajax

support in UI components as a built-in feature.