# BSF4ooRexx
# Apache Tomcat - Cookbook

Burgstaller Andreas

Matriculation No.: 11720941

Winter Term 2020/21

**Course:** Seminar BIS

**Course-No.:** 0100

**SBWL:** Business Information Systems

**Instructor:** ao.Univ.Prof. Dr. Rony G. Flatscher

## Declaration of Authorship

I, Andreas Burgstaller, hereby declare that

1. I have written this seminar paper independently and without the aid of unfair or unauthorized resources. Whenever content was taken directly or indirectly from other sources, this has been indicated and the source referenced.

2. this seminar paper has neither previously been presented for assessment, nor has it been published.

3. this seminar paper is identical with the assessed paper and the paper which has been submitted in electronic form.

Date: 17.12.2020

Signature:

## Table of Contents

# 1 Introduction

## 1.1 Overview

This seminar paper was written during the seminar of the SBWL Business Information Systems in winter term 2020/21. The objective of this paper is to provide an introduction how to use Apache Tomcat with the programming language Object Rexx. Additionally, the extension BSF4ooRexx is used.

To begin, there will be a technical background about the used technologies and instructions for their installation. Thereafter different nutshell examples will be shown. We start with basic examples of JSP and Servlets and continue with more advance and more practical examples. Last, we will go over the JSTL Standard Tag Library and the most important functions you need to know to create and maintain a database connection.

## 1.2 Apache Tomcat

Apache is one of the most popular open-source webserver software packages and enables websites to share their data in the web since 1995. Therefore, it is one of the oldest and most reliable webservers, which can be used free of charge. Apache Tomcat is an open-source webserver and web container, that enables the deployment of Java based web applications. The implementation supports the use of Java Servlets and Java Server Pages, thus granting the user the opportunity to create and provide dynamic web content. This is the main difference compared to the classic apache webserver approach, which is designed for static content like HTML, pictures, audio or text files. [1]

## 1.3 Open Object Rexx

The presented nutshell examples in this paper are based on the script language Object Rexx. This is a fully functional programming language, with the focus on simplicity and human readable code syntax. Moreover, it supports the common concept of object orientated programming. Additionally, an open-source project by the Rexx Language Association called Open Object Rexx is used. It provides a simple implementation of the script language Object Rexx.

## 1.4 BSF4ooRexx

Additional to Open Object Rexx, the framework BSFooRexx (Bean Scripting Framework for ooRexx) is used for the development of the nutshell examples. It enables the interaction with the Java Runtime Environment, as well as the implementation of Java classes and methods. Moreover, it allows to camouflage the oRexx code as Java code and therefore grants the access to all Java compatible technologies.

„Everything that is available in Java becomes directly available to ooRexx“[2] (Flatscher, 2016)

# 2 Installation

## 2.1 Installation ooRexx

Before beginning, it is important to clarify which system we are using. Luckily, Open Object Rexx is available in 32-bit and 64-bit for Windows, MacOS and Linux. To download the latest ooRexx Version go to sourceforge.net/projects/oorexx/files/oorexx/

For the nutshell examples we will be using Version 5.0 for Windows (64-bit). After executing the installation file and following all steps, ooRexx is successfully installed. For all following installations we will stick to the 64-bit Version. Using different versions can lead to unnecessary errors and should always be avoided.

Next step is to check if the right Java version is installed. In Windows you can easily do this by using the command "java -version" in your control panel. The same command can be used in the Linux or MacOS terminal. If the required version is installed, continue with the next step. If the version differs or no Version is installed, it is recommended to download the right version directly from official Java website.

## 2.2 Installation BSF4ooRexx

Visit sourceforge.net/projects/bsf4oorexx/files/ and choose the newest version of BSF4ooRexx. For the nutshell examples we will be using "v641-20201124-beta", but do not be afraid to use a different version. BSF4ooRexx is in constant development and therefore it does not take long until a new version appears.

Follow the following steps for the installation:[3]

1. Download the Windows version of BSF4ooRexx
2. Unzip the download archive.
3. Next navigate to the subdirectory "bsf4oorexx\install\windows"
4. Execute the install.cmd file
5. Wait for the installation to finish

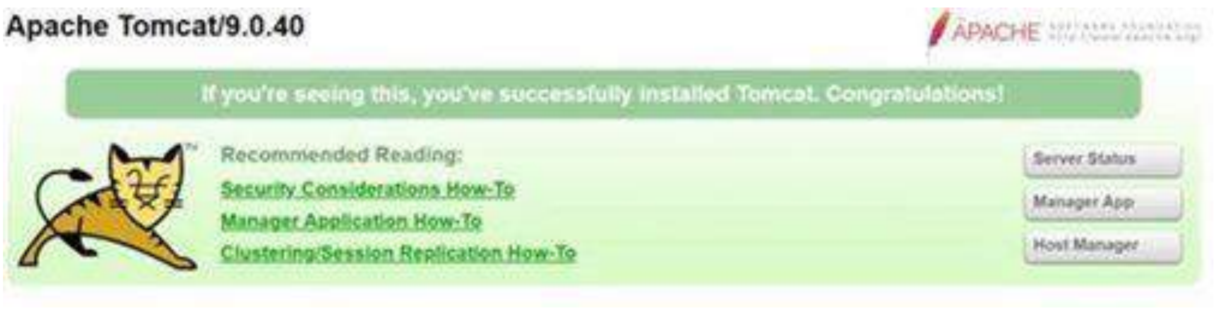
During the installation, information is printed in the terminal window. It is useful to track the status of the installation and to recognize problems that occur during the installation process. After the script is finished, you will be informed if the installation was successful. The complete log file is saved in the home directory as "BSF4ooRexx_Date.log". If facing any errors after the installation, it is recommended to first check the log files. After this, the installation is complete and the BSF4ooRexx Framework is ready to use. Additional environment variables can be added for the BSF4ooRexx installation folder.

This can resolve errors when the program does not find the folder where BSF4ooRexx is located. The same applies to the Java installation folder.

## 2.3 Apache Tomcat

### Installation

In this paper Tomcat 9 is used, which is available at [tomcat.apache.org/download.](tomcat.apache.org/download) Download the “32-bit/64-bit Windows Service Installer” and execute the Installation Wizard. For the following nutshell examples, it is not necessary to change any values during the installation. It is sufficient to follow the steps without changing any input values. Now start the “Tomcat9.exe” and enter [localhost:8080](localhost:8080) in the Browser. If the Installation was successful, a webpage with useful information about Tomcat 9 and further possible configurations will open.



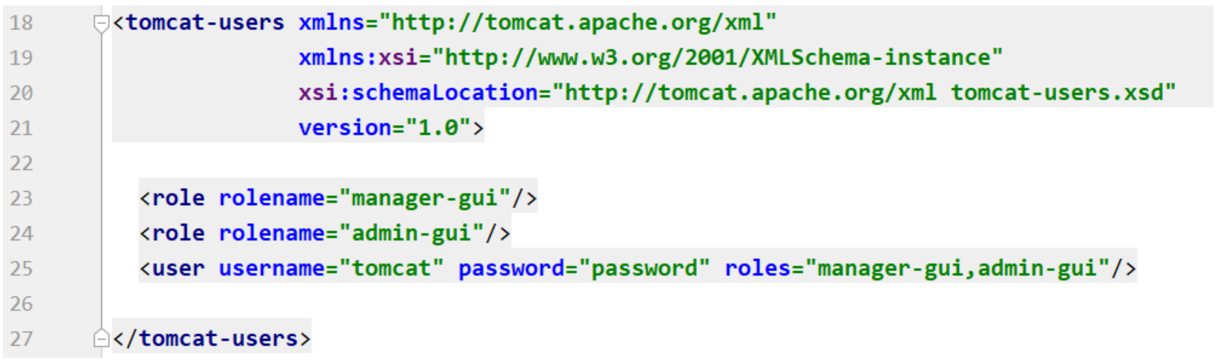
Apache Tomcat/9.0.40

If you're seeing this, you've successfully installed Tomcat. Congratulations!

APACHE
Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To
Server Status
Manager App
Host Manager

Figure 1: Apache Tomcat - Start Page

The next step is to create a Tomcat-Manager-Account, to access the Manager-Interface. Open “conf/tomcat-users.xml” in the installation folder of Tomcat and add the following code. The fastest way to do this is to open the file with a text editor. Then add the roles “manager-gui” and “admin-gui”. These roles grant a user the necessary rights needed and by simply creating a new user and defining his roles, everything is ready.[4]



```
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">

  <role rolename="manager-gui"/>
  <role rolename="admin-gui"/>
  <user username="tomcat" password="password" roles="manager-gui,admin-gui"/>

</tomcat-users>
```

Figure 2: Tomcat9.0/conf/tomcat-user.xml

Now the Manager-Interface can be accessed with the newly created user. Start the Tomcat server and open localhost:8080/manager in the Browser. Then enter username and password in the login window. If the user was created correctly, it will load a directory with all available applications in the "Tomcat9.0/webapp" folder.

## Testing ooRexx and BSF4ooRexx

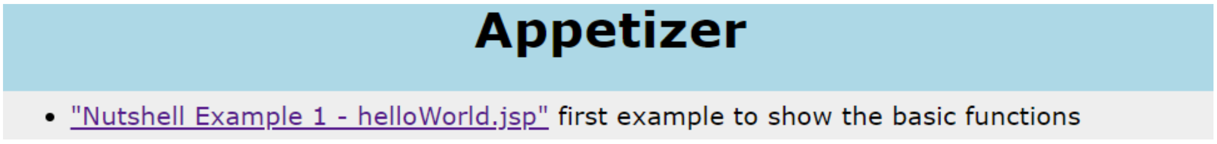
First it is important to provide Apache Tomcat with the necessary resources. To do that, add the following libraries in the "Tomcat 9.0\lib" folder of the Tomcat installation.

| | |
|---|---|
| **javax.ScriptTagLibs.jar** | contains BSF and JSR-223 tag libraries |
| **bsf4ooRexx-v641-20201124.jar** | contains the Bean Scripting Framework that acts as bridge between Java and ooRexx |

The "bsf4ooRexx-v641-20201124.jar" can be found in the installation folder of the BSF4ooRexx installation. The "javax.ScriptTagLibs.jar" and the "javax.demoRexx.war" example can be downloaded from sourceforge. The demo application is perfect to verify if everything is working correctly. After adding the libraries, add the "javax.demoRexx.war" to the "Tomcat 9.0\webapps" folder and unzip the file. The webapps directory is where all web applications are deployed.

Finally, it is time to start the Tomcat server and login to the Manager-Interface. If all step were executed correctly, "java.demoRexx20201124" should be listed. By clicking on the entry, it switches to an overview of all the JSP-Files the example contains. If everything works correctly they demonstrate the most important functions of JSP and oRexx and give a good first impression of the following nutshell examples.

# 3 Appetizer



Appetizer

- "Nutshell Example 1 - helloWorld.jsp" first example to show the basic functions

Figure 3: localhost/Appetizer/index.html

## 3.1 Nutshell Example 1 - Hello World (Basic JSP Example)

The first nutshell example is a simple JSP page, that prints the historic sentence "Hello World". Its purpose is to give an overview of the file structure of applications in Apache Tomcat and to show the basics of JSP. The applications in Adobe Tomcat are stored in the webapps folder.

| | |
|---|---|
| **/WEB-INF** | invisible for the web user and contains all resources needed to run the application e.g., JAR files, libraries, tag libraries |
| **script-bsf.tld** | describes the BSF tag library |
| **web.xml** | deployment descriptor of the web application e.g., URL Mappings, Servlet definition |
| **index.html** | default/start page of the application |
| **HelloWorld.jsp** | JSP file that prints Hello World with the current date and time |

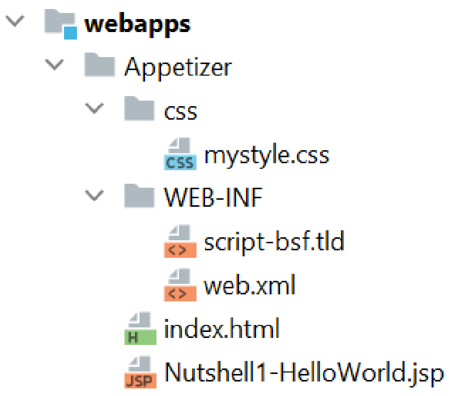


Figure 4: webapps folder

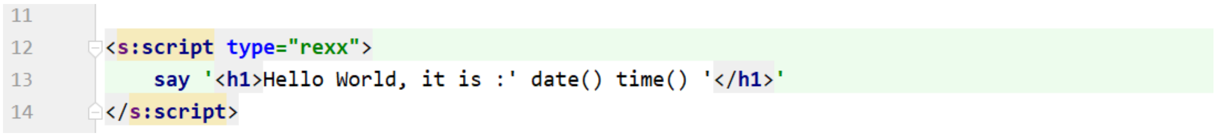
JSP pages are server generated web pages, that contain Java code. The webserver parses the code and generates a HTML that is sent to the user. The nutshell examples are using the BSF tag library to allow the use of oRexx code. Therefore, it is not necessary to use any Java code in these examples. To design webpages in JSP use HTML tags. It is common practice to create a base structure in HTML and then add scripts that create the dynamic content. [5]

To use the tag library in a JSP file, add the line from Figure 5 outside of the <html> tag.

```
<%@ taglib uri="/WEB-INF/script-bsf.tld" prefix="s" %>
```

Figure 5: Nutshell1-HelloWorld.jsp – tag library definition

With the support of the BSF library, add a script tag in the <body> tag.

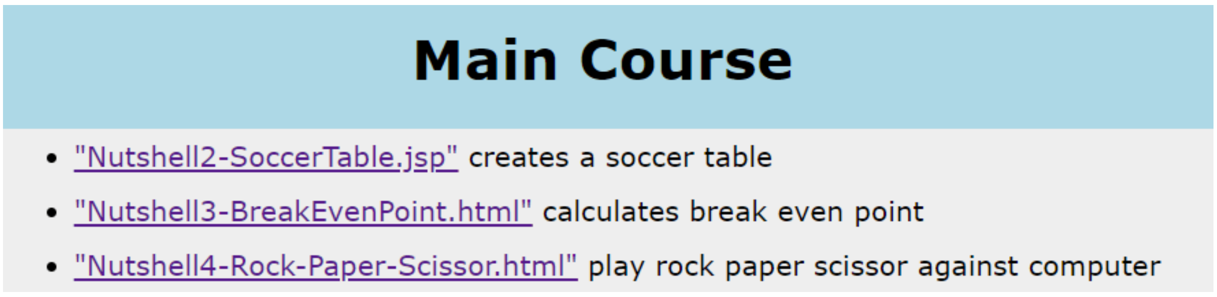


```

<s:script type="rexx">
    say '<h1>Hello World, it is :' date() time() '</h1>'
</s:script>
```

Figure 6: Nutshell1-HelloWorld.jsp – oRexx script

In the script tag it is possible to use all functions of oRexx and by simply using the "say" method, code is added to the HTML. Furthermore, is it feasible to add a reference for a CSS file, to improve the visual appearance of the application. Last, add a reference that points to the Nutshell1-HelloWorld.jsp into the index.html. This page acts as homepage of the application and is loaded when localhost:8080/Appetizer/ is requested.

# 4 Main Course



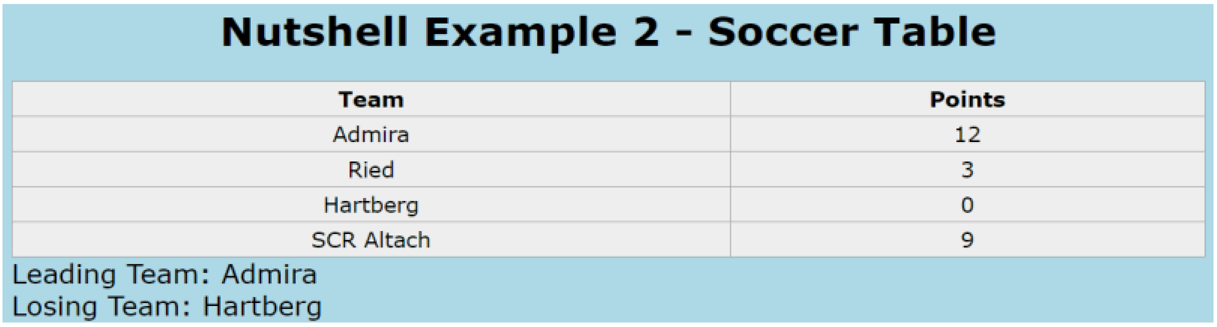
Main Course

- "Nutshell2-SoccerTable.jsp" creates a soccer table
- "Nutshell3-BreakEvenPoint.html" calculates break even point
- "Nutshell4-Rock-Paper-Scissor.html" play rock paper scissor against computer

Figure 7: localhost:8080/Main%20Course/index.html

## 4.1 Nutshell Example 2 - Soccer Table (Advanced JSP Example)

The second nutshell example is an advanced form of the previous one. The main difference is that it uses more complex oRexx code e.g., arrays and loops. The oRexx code creates an array of soccer teams and their individual points, which is filled with static test data. Then a loop iterates over this array and for each entry a row is added to an HTML table. Additionally, the loop compares the points of the teams. At the bottom of the table the best and worst team is listed.



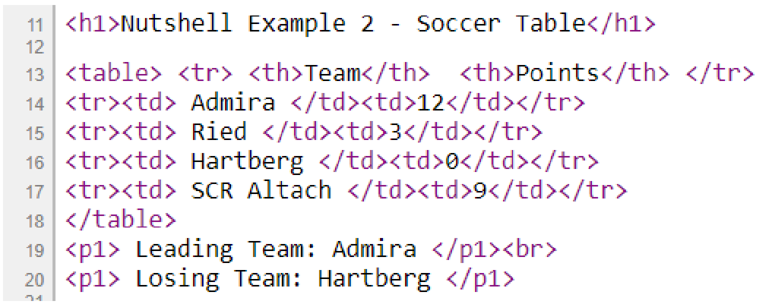
Nutshell Example 2 - Soccer Table

| Team | Points |
|---|---|
| Admira | 12 |
| Ried | 3 |
| Hartberg | 0 |
| SCR Altach | 9 |

Leading Team: Admira
Losing Team: Hartberg

Figure 8: localhost:8080/Main%20Course/Nutshell2-SoccerTable.jsp

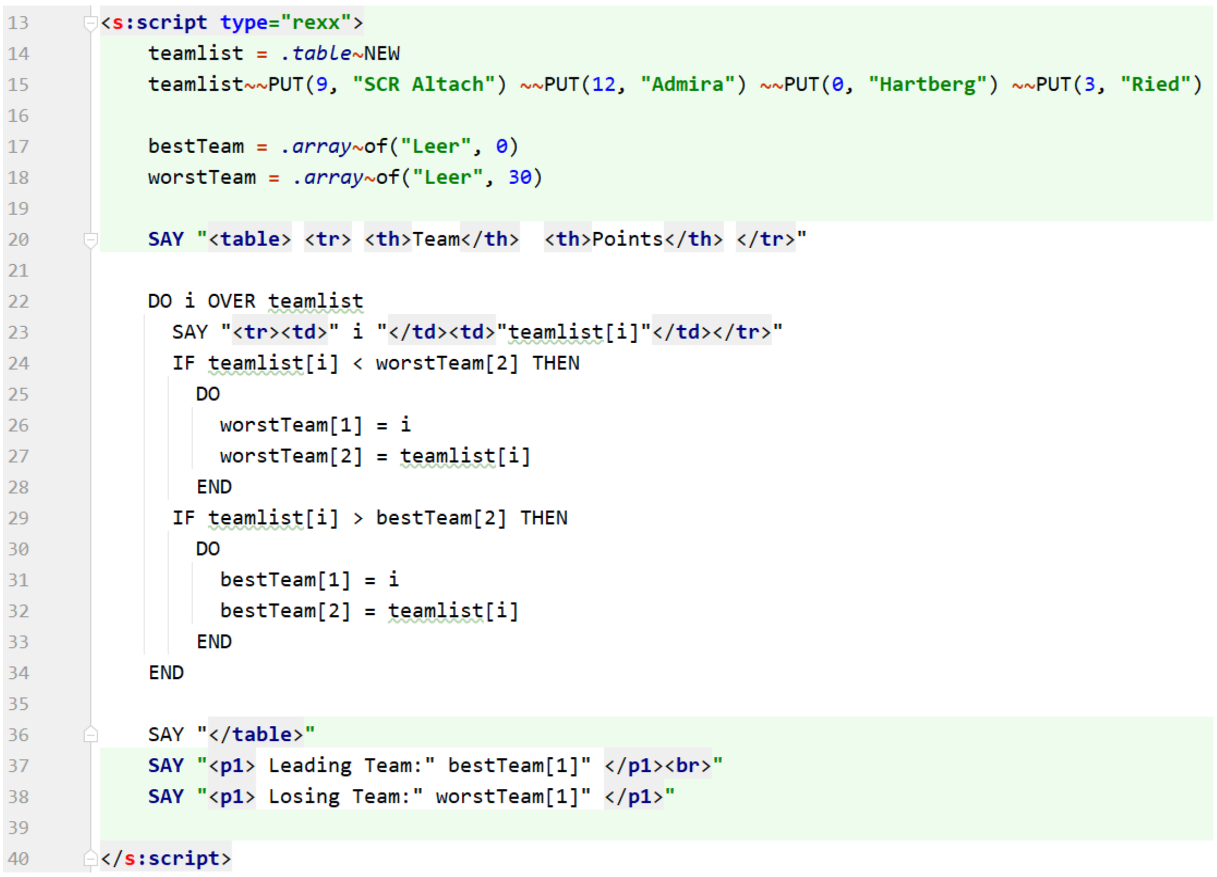
The complete HTML can be created in the script tag. Therefore, it is very user-friendly to create dynamic content. Figure 9 shows the HTML content that is created by the second nutshell example.



```
<h1>Nutshell Example 2 - Soccer Table</h1>

<table> <tr> <th>Team</th>  <th>Points</th> </tr>
<tr><td> Admira </td><td>12</td></tr>
<tr><td> Ried </td><td>3</td></tr>
<tr><td> Hartberg </td><td>0</td></tr>
<tr><td> SCR Altach </td><td>9</td></tr>
</table>
<p1> Leading Team: Admira </p1><br>
<p1> Losing Team: Hartberg </p1>
```

Figure 9: Nutshell2-SoccerTable.jsp - generated HTML content

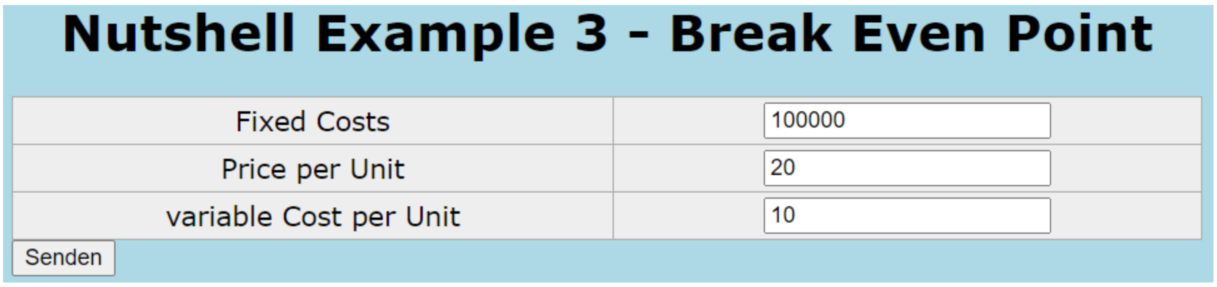
Figure 10 shows the complete code that is used in this nutshell example. It illustrates that nearly every HTML content can be created in the script tag, by simply using the say method. In combination with the other functionalities of the oRexx language this becomes a very powerful tool.



```
<s:script type="rexx">
    teamlist = .table~NEW
    teamlist~~PUT(9, "SCR Altach") ~~PUT(12, "Admira") ~~PUT(0, "Hartberg") ~~PUT(3, "Ried")

    bestTeam = .array~of("Leer", 0)
    worstTeam = .array~of("Leer", 30)

    SAY "<table> <tr> <th>Team</th>  <th>Points</th> </tr>"

    DO i OVER teamlist
      SAY "<tr><td>" i "</td><td>"teamlist[i]"</td></tr>"
      IF teamlist[i] < worstTeam[2] THEN
        DO
          worstTeam[1] = i
          worstTeam[2] = teamlist[i]
        END
      IF teamlist[i] > bestTeam[2] THEN
        DO
          bestTeam[1] = i
          bestTeam[2] = teamlist[i]
        END
    END

    SAY "</table>"
    SAY "<p1> Leading Team:" bestTeam[1]" </p1><br>"
    SAY "<p1> Losing Team:" worstTeam[1]" </p1>"

</s:script>
```

Figure 10: Nutshell2-SoccerTable - oRexx script

## 4.2 Nutshell Example 3 - Break-Even-Point (GET Method)



**Nutshell Example 3 - Break Even Point**

| | |
|---|---|
| Fixed Costs | 100000 |
| Price per Unit | 20 |
| variable Cost per Unit | 10 |

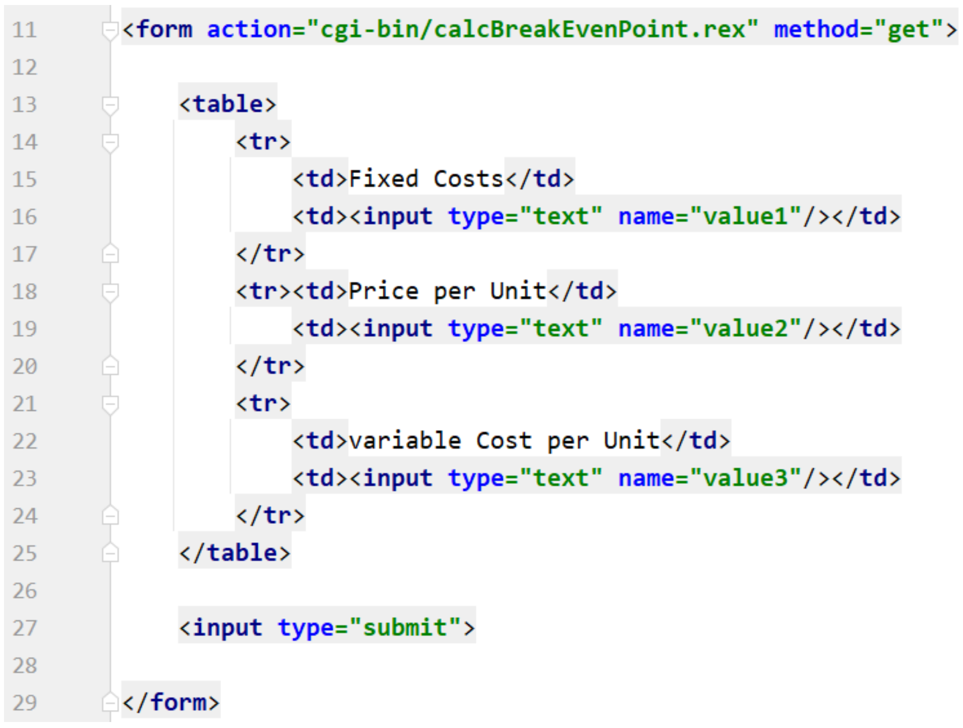
Figure 11: localhost:8080/Main%20Course/Nutshell3-BreakEvenPoint.html

The third nutshell example demonstrates how to create a calculator for the calculation of the break-even-point. To catch the input values, it is recommended to use the <form> element. This is the easiest way to collect user data and send it to the web server, where the information is processed. In this example the GET method will be used to transport the data. With this method the values get appended to the page request.

For each input, a name-value pair is added and additionally the pairs are separated by a "&" sign. The values and the page request are separated by an "?" sign.

Example Request: http://www.test.com/index.htm**?**name1=value1**&**name2=value2[6]

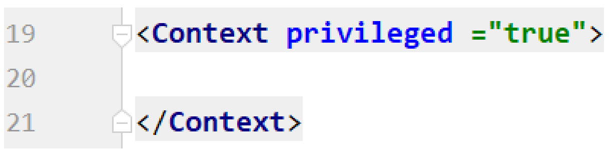
Figure 12 shows the complete <form> element of the break-even-point nutshell example. The first step for creating a <form> element is to define its action. This action is carried out when the page gets submitted, usually this is done by a button. In this case an oRexx servlet is executed and prints the calculated break-even-point to the screen. To collect the values, an <input> element with the type “text” is used. This creates a blank single-line text input field. To improve the format of the webpage, the elements are put in a <table> element.



```
<form action="cgi-bin/calcBreakEvenPoint.rex" method="get">

    <table>
        <tr>
            <td>Fixed Costs</td>
            <td><input type="text" name="value1"/></td>
        </tr>
        <tr><td>Price per Unit</td>
            <td><input type="text" name="value2"/></td>
        </tr>
        <tr>
            <td>variable Cost per Unit</td>
            <td><input type="text" name="value3"/></td>
        </tr>
    </table>

    <input type="submit">

</form>
```

Figure 12: Nutshell3-BreakEvenPoint.html - <form> element

To use the calcBreakEvenPoint.rex servlet, that is defined as action in the <form> element, it is necessary to implement CGI (Common Gateway Interface) support. This interface enables the web server to interact with external content-generating programs. By default, CGI support is disabled in Tomcat and thereby some changes are needed. First go to Tomcat9.0/conf/context.xml and open the file. There add the privileged attribute to the <Context> tag and set it true.[7]



```
<Context privileged ="true">

</Context>
```

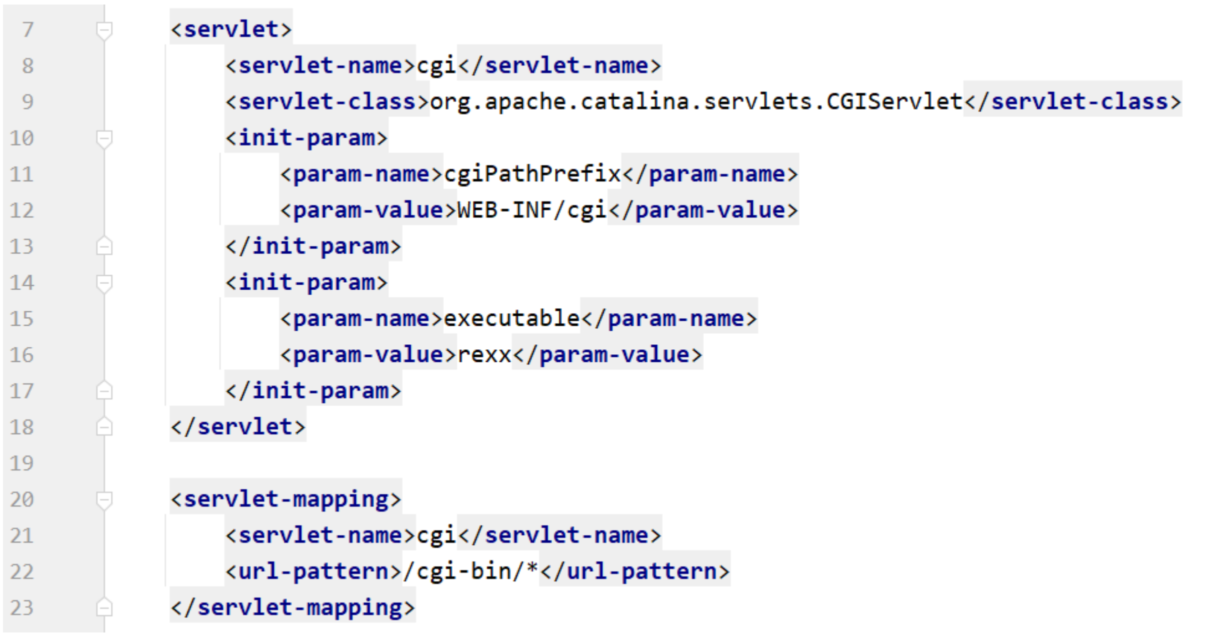
Figure 13: Tomcat9.0/conf/context.xml – privileged = “true”

Modifying the context.xml in this directory has the drawback, that the changes apply for all web applications. This could lead to security issues and it is recommended to create a context.xml that applies only for one web application. To do this, the file must be placed in the WEB-INF folder of the web application and a root.xml file must be placed in the Tomcat9.0/conf/Catalina/localhost directory that points to the specific web application.

```
<Context
  docBase="<yourApp>"
  path=""
  reloadable="true"
/>
```

**Figure 14: Tomcat9.0/conf/Catalina/localhost/root.xml**

The second step to add CGI support is to adapt the web.xml in the WEB-INF folder. Add a servlet definition, that implements the CGIServlet class and define the storage location of the servlets as WEB-INF/cgi. Then create the associated folder, name it "cgi", in the WEB-INF directory. Last, add an appropriate servlet-mapping, which is traditionally the URL /cgi-bin/*.

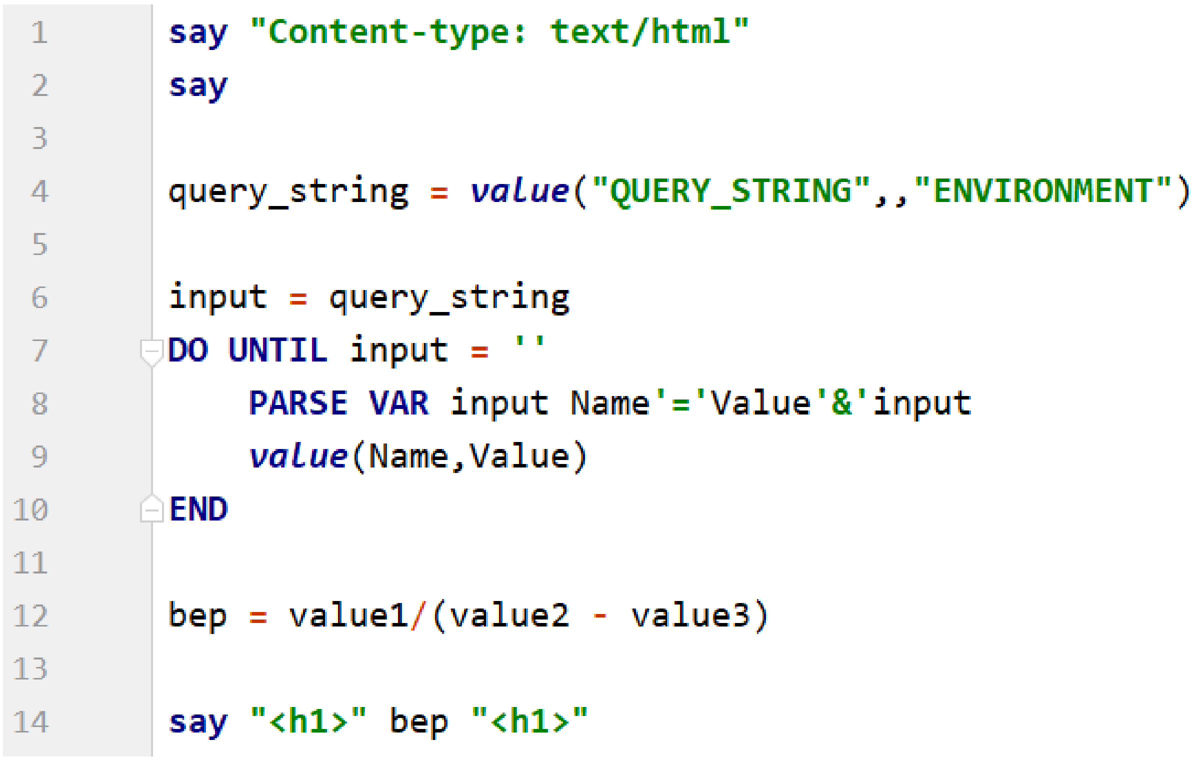


```
<servlet>
    <servlet-name>cgi</servlet-name>
    <servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>
    <init-param>
        <param-name>cgiPathPrefix</param-name>
        <param-value>WEB-INF/cgi</param-value>
    </init-param>
    <init-param>
        <param-name>executable</param-name>
        <param-value>rexx</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>cgi</servlet-name>
    <url-pattern>/cgi-bin/*</url-pattern>
</servlet-mapping>
```

**Figure 15: MainCourse/WEB-INF/web.xml – servlet definition**

Finally, it is possible to execute the oRexx Servlets by the GET method of the HTML <form>. Figure 16 shows the calcBreakEvenPoint.rex servlet. Just like in previous nutshell examples, the HTML code is being created by using the "say" method. Since the GET method directs to a different webpage, it is necessary to create a new HTML file from scratch.

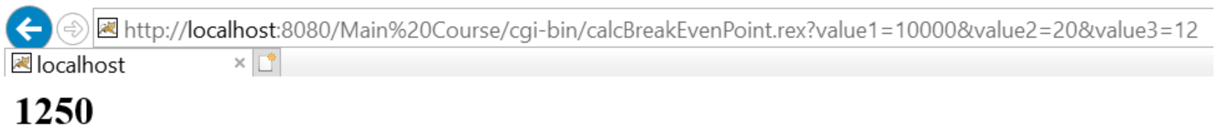
First the content type must be defined and set to text/html. Next, get the values that were appended to the page request. Then use a loop to extract every key and value from the string. The name of each value was defined in the <input> tag before. The last step is to insert the values into the formular to calculate the break-even-point and print the solution to the screen. This simple formular was chosen to demonstrate how to implement oRexx servlets and execute them with the GET method. Instead of this simple calculation, any other oRexx code could be processed.



```
say "Content-type: text/html"
say

query_string = value("QUERY_STRING",,"ENVIRONMENT")

input = query_string
DO UNTIL input = ''
    PARSE VAR input Name'='Value'&'input
    value(Name,Value)
END

bep = value1/(value2 - value3)

say "<h1>" bep "<h1>"
```

**Figure 16: /WEB-INF/cgi/calcBreakEvenPoint.rex**

To conclude on this example, developing complex web application is already possible with the content shown so far in the nutshell examples. Nevertheless, the GET method has some major disadvantages that limits its practicable uses. The core problem is that the GET method appends the values in plain text. This is a major security risk and thereby makes it unusable for sensitive data. Moreover, there are some data size and type restrictions. Luckily, there exists another method that can send information to the web server. This is the POST method, which will be used in the next nutshell example.



```
http://localhost:8080/Main%20Course/cgi-bin/calcBreakEvenPoint.rex?value1=10000&value2=20&value3=12
localhost

1250
```

**Figure 17:: Nutshell3-BreakEvenPoint.html – GET request and servlet output**

## 4.3 Nutshell Example 4 – Rock-Paper-Scissor (POST Method)

The fourth nutshell example will demonstrate how to use the POST method to execute a more complex oRexx servlet code example. The basic function of this example is to play a rock-paper-scissor game against a randomly choosing computer enemy. The example should also illustrate how to implement image files or other resources in HTML pages.

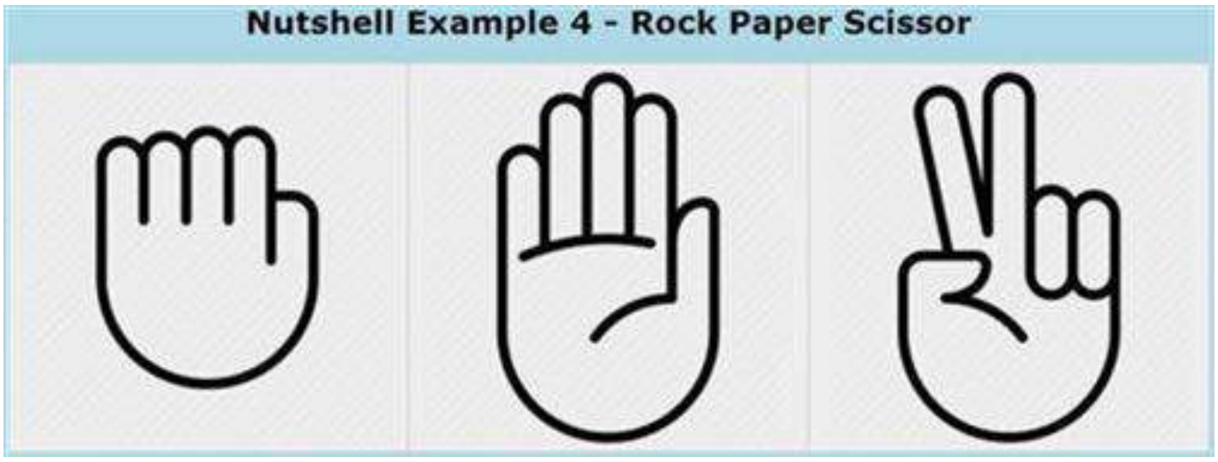


Figure 18: localhost:8080/Main%20Course/Nutshell4-Rock-Paper-Scissor.html

Figure 19 shows the complete <form> element of the rock-paper-scissor nutshell example. We use a similar <form> element as in the previous nutshell examples. This time three <input> elements are used which all act as submit buttons. Every element has an .png file linked, that shows either an image of rock, paper or scissor. The files are stored in the web application in a folder named images. When an element gets clicked the HTML becomes submitted and a POST method executes the rockpaperscissor.rexx servlet.

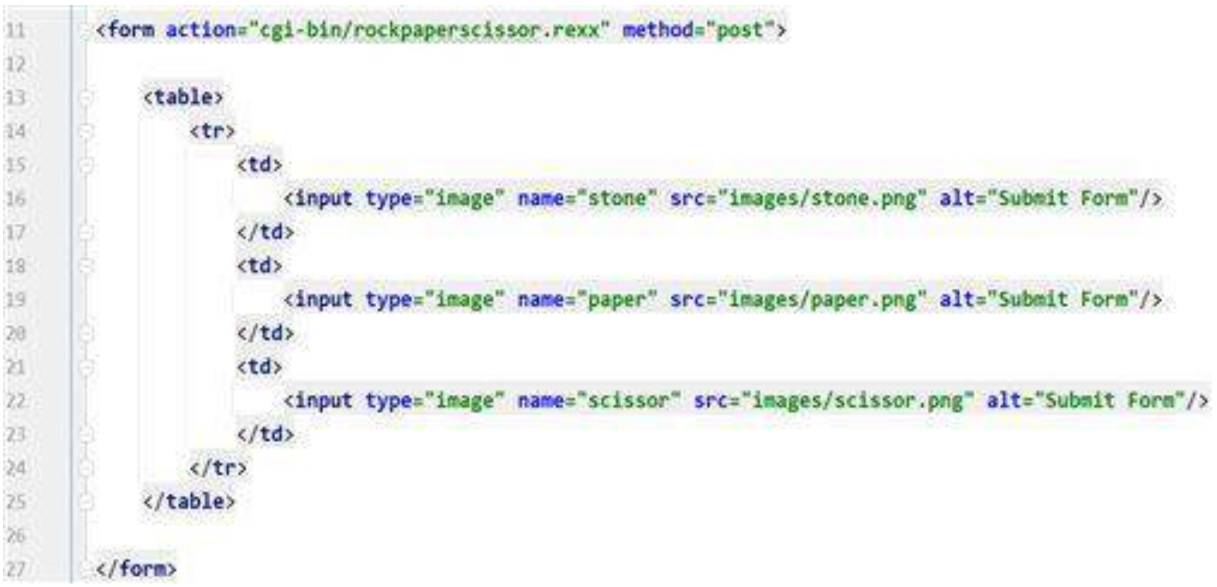

```
<form action="cgi-bin/rockpaperscissor.rexx" method="post">

    <table>
        <tr>
            <td>
                <input type="image" name="stone" src="images/stone.png" alt="Submit Form"/>
            </td>
            <td>
                <input type="image" name="paper" src="images/paper.png" alt="Submit Form"/>
            </td>
            <td>
                <input type="image" name="scissor" src="images/scissor.png" alt="Submit Form"/>
            </td>
        </tr>
    </table>

</form>
```

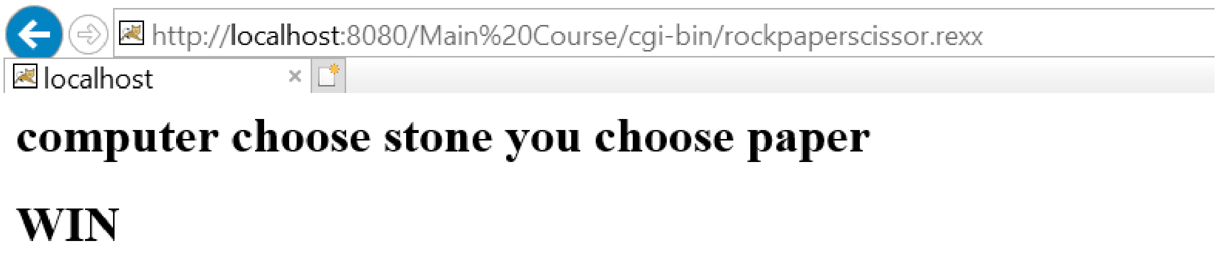
Figure 19:Nutshell4-Rock-Paper-Scissor.html - <form> element

Figure 20 shows the oRexx servlet which reads the submitted values. Like before, the definition of the content-type is the first step. This time the extraction of the values differs with the POST method. The fastest way to get the right data is to look at the complete information that is stored in the content_body. In this example, we know that our variable is at the front and is either “rock”, “paper” or “scissor”. By using the PARSE VAR method, we can split the information and get the necessary value.

Then a computer input is randomly generated with the RANDOM method. Last the input of the computer and the user get compared and a winner is decided. This is done by using IF-ELSE-IF statements.

```
say "Content-type: text/html"
say

Parse Pull content_body
PARSE VAR content_body user_input'.'message

values = .array~of("stone","paper","scissor")
comp_input = values[RANDOM(1,3)]

say "<h1> computer choose " comp_input " you choose " user_input "</h1>"

if user_input = comp_input then
    SAY "<h1>"DRAW"</h1>"
else if user_input = "scissor"then
    if comp_input = "stone" then
        SAY "<h1>"LOSE"</h1>"
    else
        SAY "<h1>"WIN"</h1>"
else if user_input = "paper"then
    if comp_input = "scissor" then
        SAY "<h1>"LOSE"</h1>"
    else
        SAY "<h1>"WIN"</h1>"
else if user_input = "stone"then
    if comp_input = "paper" then
        SAY "<h1>"LOSE"</h1>"
    else
        SAY "<h1>"WIN"</h1>"
```

Figure 20: /WEB-INF/cgi/rockpaperscissor.rexx


http://localhost:8080/Main%20Course/cgi-bin/rockpaperscissor.rexx
localhost

computer choose stone you choose paper

WIN

Figure 21: Nutshell4-Rock-Paper-Scissor.html - output of rockpaperscissor.rexx servlet

The POST method, in contrast to the GET method, transfers the information via HTTP headers. The security of this method depends on the used HTTP protocol. HTTPS ensured that the information is encrypted and secure. Another advantage is that the POST method has no data size restrictions. It is also possible to send in ASCII which is not possible with the GET method.

# 5 Side Dish

This chapter is about features which have been used in all the nutshell examples but were not further explained. The following information is practically oriented and will facilitate the development of web applications.

## 5.1 Cascading Style Sheets

CSS is a feature to define how HTML elements are displayed. This could be the background color, layout, font-size, font-family, etc. The syntax rules are very simple. Basically, you use a selector that points to the HTML element and add declarations of the properties with the desired values. [8]

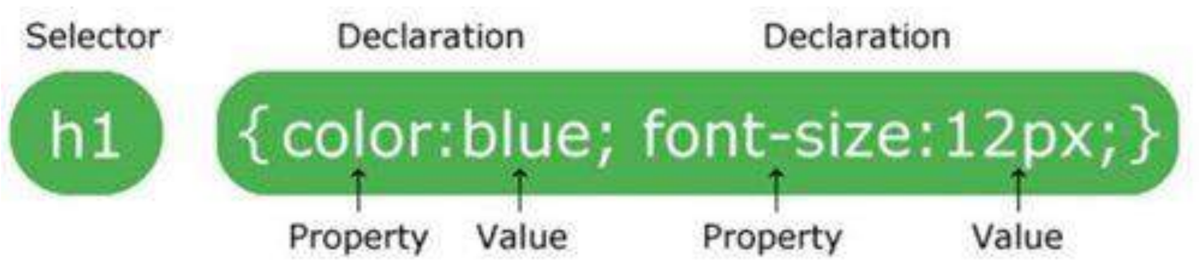
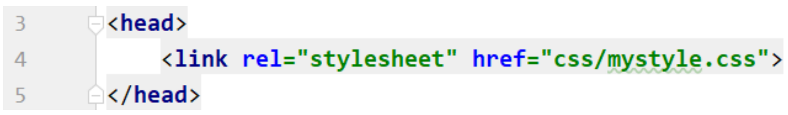


Figure 22: https://www.w3schools.com/css/css_syntax.asp

There are three different ways to use stylesheets in HTML.

### External CSS

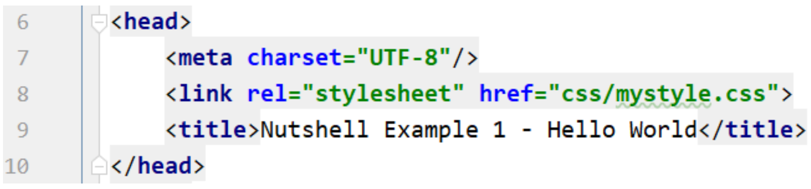
When using external CSS files, the stylesheet is stored in an individual file. This has the advantage, that by changing only one file, the complete style of an HTML can be changed. To include the stylesheet reference, add a <link> element in the header and define the path of the external CSS file. The referenced files must be stored in the web application, usually in a folder named css.



```
<head>
    <link rel="stylesheet" href="css/mystyle.css">
</head>
```

Figure 23: including external CSS stylesheets inside the header

This method was primary used in the nutshell examples and is recommended for usual development.



```
<head>
    <meta charset="UTF-8"/>
    <link rel="stylesheet" href="css/mystyle.css">
    <title>Nutshell Example 1 - Hello World</title>
</head>
```

Figure 24: Nutshell1-HelloWorld.jsp – header definition

### Internal CSS

Internal style sheets can be useful, if one single HTML page has a unique style. The stylesheet is defined inside the HTML body by using the <style> element.

```
<body>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</body>
```

Figure 25: including internal stylesheet in the body

## Inline CSS

Inline style can be useful if the style should only apply to one single element. Use it by adding a style attribute to the relevant element.

```
<h1 style="color:blue;text-align:center;">This heading is blue and centered</h1>
<p style="color:red;">This paragraph is red</p>
```

Figure 26: definition of styles for specific elements

## CSS Templates

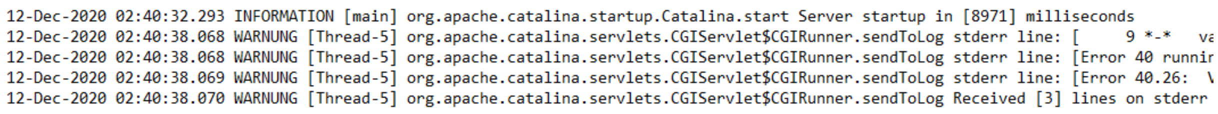
Creating an advanced stylesheet with the right layout and fancy design can be a real challenge. An alternative to creating own stylesheets from scratch is using resources from the internet. There are many different templets that are completely free to save, modify and share. These stylesheets often include responsive elements which enhance the usability and appearance of the webpages.

Figure 27: https://www.w3schools.com/w3css/w3css_templates.asp

Before using a templet for commercial use, always inspect the copyright notices. There are many templates available online without any ownership claims. Some templates require to visible credit the owner somewhere on the webpage. Usually this is done in the footer of the HTML.

## 5.2 Tips for Error Handling

In the event of inexplicable errors, it is recommendable to check the Apache Tomcat logging files. By default, they are stored in the “log” folder of the Tomcat installation. The most noteworthy for the development is the tomcat9-stderr.”date”.log. It records the exceptions that occur while the server is running and is a useful information source when running into problems.



```
12-Dec-2020 02:40:32.293 INFORMATION [main] org.apache.catalina.startup.Catalina.start Server startup in [8971] milliseconds
12-Dec-2020 02:40:38.068 WARNUNG [Thread-5] org.apache.catalina.servlets.CGIServlet$CGIRunner.sendToLog stderr line: [     9 *-*   va
12-Dec-2020 02:40:38.068 WARNUNG [Thread-5] org.apache.catalina.servlets.CGIServlet$CGIRunner.sendToLog stderr line: [Error 40 runnir
12-Dec-2020 02:40:38.069 WARNUNG [Thread-5] org.apache.catalina.servlets.CGIServlet$CGIRunner.sendToLog stderr line: [Error 40.26:  \
12-Dec-2020 02:40:38.070 WARNUNG [Thread-5] org.apache.catalina.servlets.CGIServlet$CGIRunner.sendToLog Received [3] lines on stderr
```

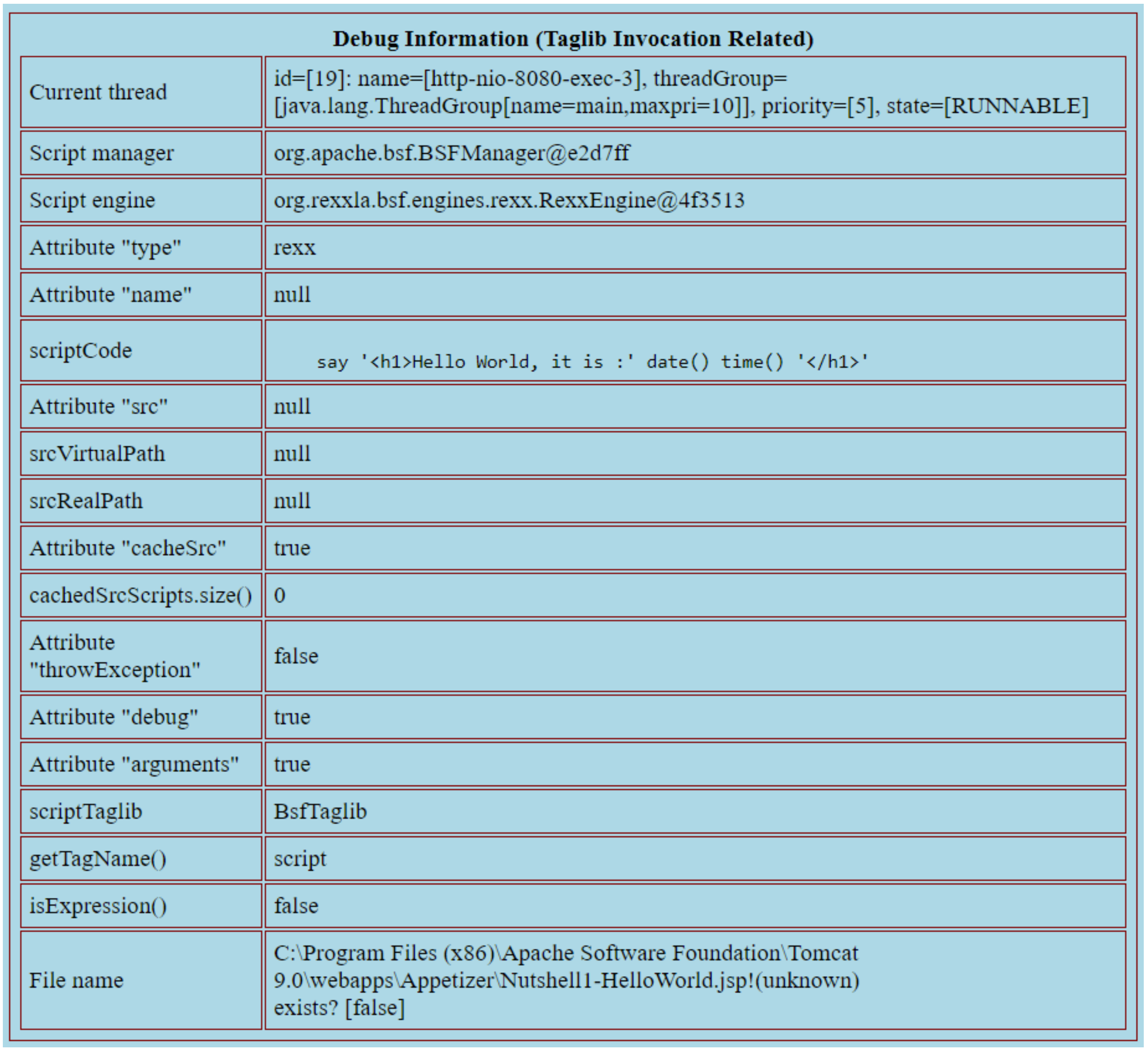
Figure 28: /Tomcat9.0/logs/tomcat9-stderr.2020-12-12

Another advise for the development is to enable the debug mode, when executing oRexx sripts. It can be done by adding the debug attribute and set it “true”.

```
<s:script type="rexx" debug ="true">
```

Figure 29: script tag - debug ="true"

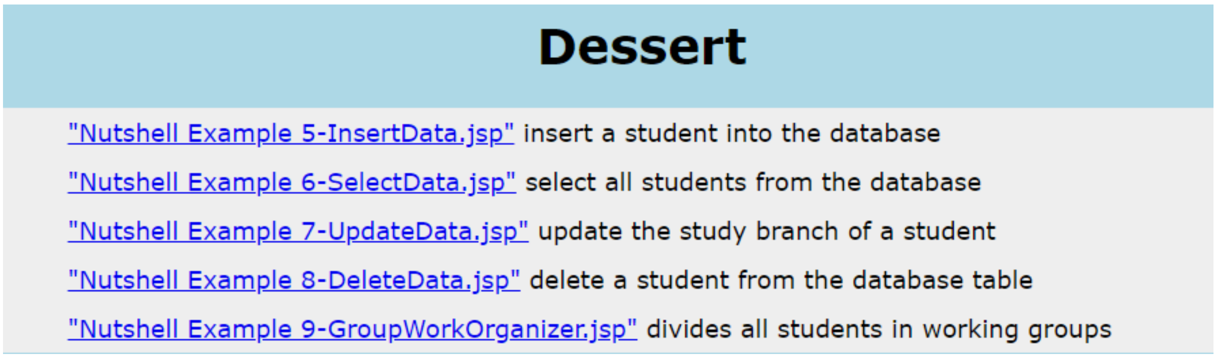
The debug mode offers an additional table of information.



| Debug Information (Taglib Invocation Related) | |
|---|---|
| Current thread | id=[19]: name=[http-nio-8080-exec-3], threadGroup=[java.lang.ThreadGroup[name=main,maxpri=10]], priority=[5], state=[RUNNABLE] |
| Script manager | org.apache.bsf.BSFManager@e2d7ff |
| Script engine | org.rexxla.bsf.engines.rexx.RexxEngine@4f3513 |
| Attribute "type" | rexx |
| Attribute "name" | null |
| scriptCode | `say '<h1>Hello World, it is :' date() time() '</h1>'` |
| Attribute "src" | null |
| srcVirtualPath | null |
| srcRealPath | null |
| Attribute "cacheSrc" | true |
| cachedSrcScripts.size() | 0 |
| Attribute "throwException" | false |
| Attribute "debug" | true |
| Attribute "arguments" | true |
| scriptTaglib | BsfTaglib |
| getTagName() | script |
| isExpression() | false |
| File name | C:\Program Files (x86)\Apache Software Foundation\Tomcat 9.0\webapps\Appetizer\Nutshell1-HelloWorld.jsp!(unknown) exists? [false] |

Figure 30: debug mode = “true” – information table

# 6 Dessert



**Dessert**

"Nutshell Example 5-InsertData.jsp" insert a student into the database

"Nutshell Example 6-SelectData.jsp" select all students from the database

"Nutshell Example 7-UpdateData.jsp" update the study branch of a student

"Nutshell Example 8-DeleteData.jsp" delete a student from the database table

"Nutshell Example 9-GroupWorkOrganizer.jsp" divides all students in working groups

Figure 31:localhost/Dessert/index.html

## 6.1 JSTL - Tag Library [9]

The nutshell examples in this chapter will demonstrate how to create a connection to a MySQL database and the execution of fundamental SQL queries. Therefore, the JSTL standard tag library will be used. It is a collection of useful JSP tags that offers support for structural tasks, XML manipulation, SQL, etc. To implement this tag library, it is necessary to add two JAR files. Download the newest version of taglibs-standard-impl-“version”.jar from tomcat.apache.org and the jstl-“version”.jar from jar-download.com. Add these JAR files into the Apache Tomcat lib folder. To include the libraries to the JSP, add the code lines of **Figure 32.** This enables the use of the <sql> and <core> tag, which will be used in later examples.

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

Figure 32: Standard Tag Library (JSTL) – definition of <core> and <sql> tag

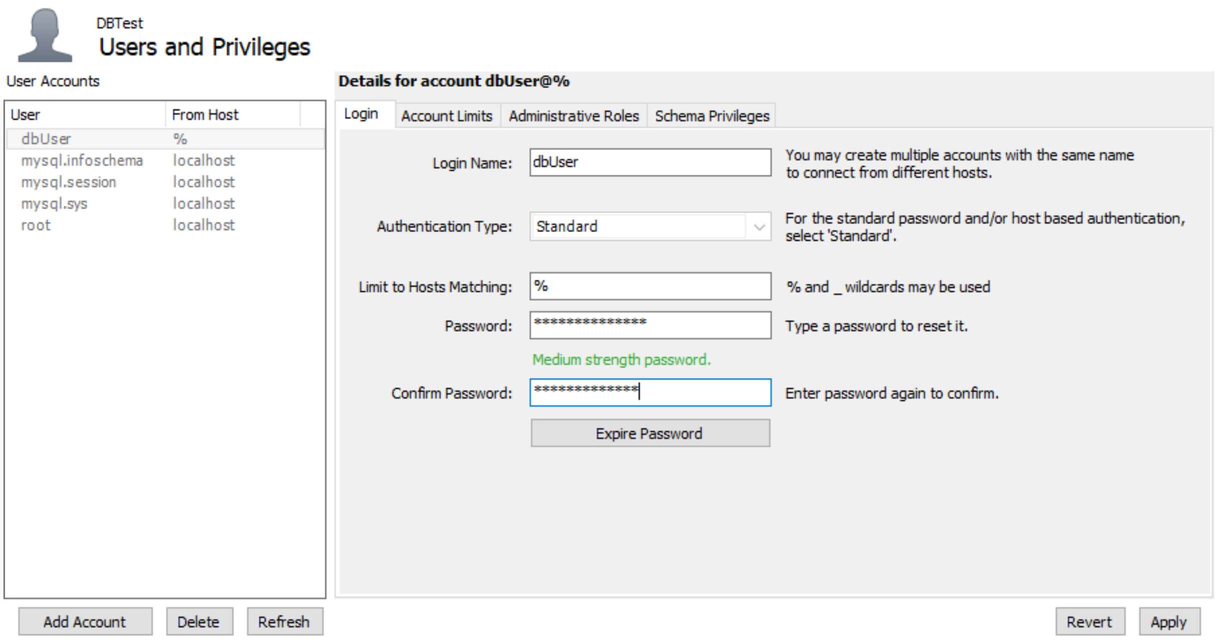
## 6.2 MySQL Database Support[10]

Before we start with the nutshell examples, it is necessary to set up a database. This can be done with the MySQL Workbench. Download the installation file from dev.mysql.com/downloads and follow the installation instruction. During the server setup of the installation a password for the root user is required. Next start MySQL Workbench and create a simple connection by clicking the “+” icon on the start page. It is not required to change any input values expect the Connection Name. Press “Test Connection” and if no errors occur the connection should work without problems. Close the window by clicking “OK”, double click the new connection and enter the root password you have chosen in the installation

Figure 33: MySQL Workbench

| | |
|---|---|
| **Connection Name:** | TestDB |
| **Hostname:** | 127.0.0.1 |
| **Port:** | 3306 |
| **Username:** | root |

Next, navigate to “Server” and “User and Privileges” in the menu bar. Click “Add Account” and enter the values for the new user. The user for the nutshell examples has the login name “dbUser” and the password “password”.



| User | From Host |
|---|---|
| dbUser | % |
| mysql.infoschema | localhost |
| mysql.session | localhost |
| mysql.sys | localhost |
| root | localhost |

Login Name: dbUser — You may create multiple accounts with the same name to connect from different hosts.

Authentication Type: Standard — For the standard password and/or host based authentication, select 'Standard'.

Limit to Hosts Matching: % — % and _ wildcards may be used

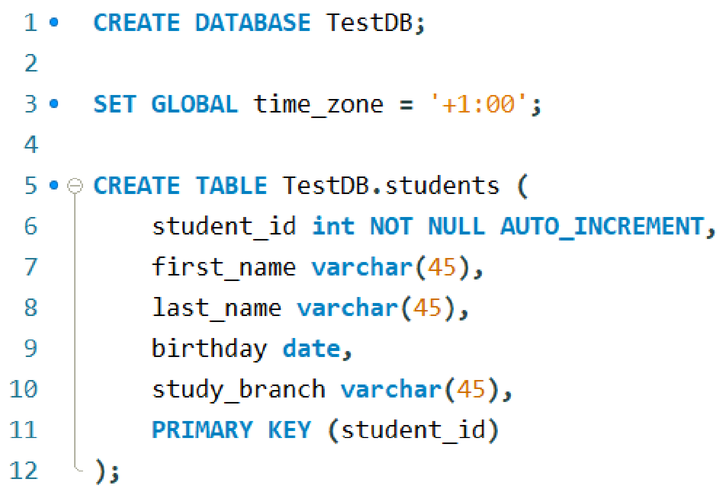
Password: *************** — Type a password to reset it.

Medium strength password.

Confirm Password: ************** — Enter password again to confirm.

Figure 34: MySQLWorkbench-> Server-> User and Privileges – add user account

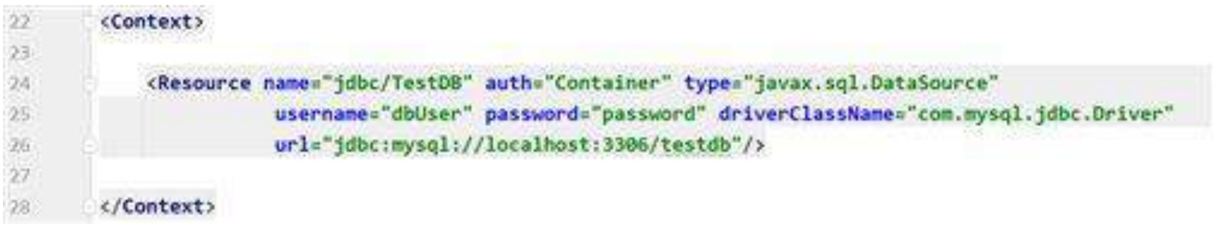
Now open a new query window in MySQL Workbench and create a database with a table that can be accessed by the nutshell examples later. The SQL query from **Figure 35** creates a database TestDB with a table for student data. Additionally, the time zone is set to avoid a common error about undefined time zones.



```sql
CREATE DATABASE TestDB;

SET GLOBAL time_zone = '+1:00';

CREATE TABLE TestDB.students (
    student_id int NOT NULL AUTO_INCREMENT,
    first_name varchar(45),
    last_name varchar(45),
    birthday date,
    study_branch varchar(45),
    PRIMARY KEY (student_id)
);
```

Figure 35: create database and table

Download the mysql-connector-java-“version”.jar from [dev.mysql.com/downloads](dev.mysql.com/downloads) and add it to the /Tomcat9.0/lib folder. The last step is to define the connection details by creating a data source in the /Tomcat9.0/conf/context.xml. Now the data base is set up and can be accessed by the web application.

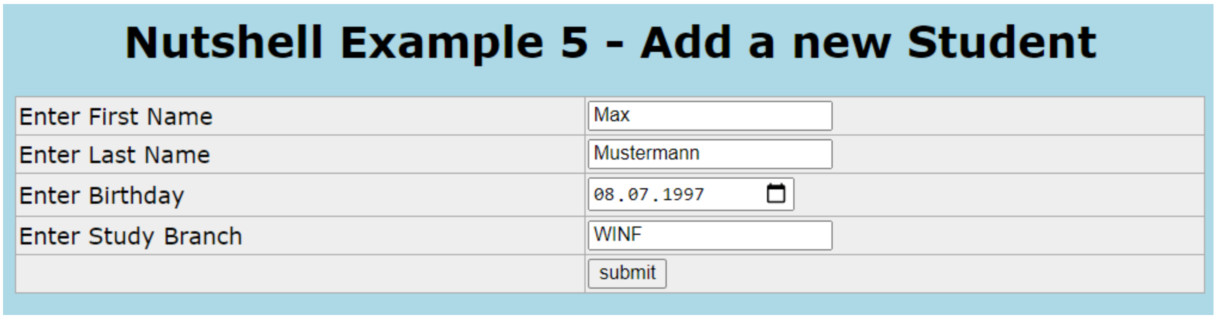


```
<Context>

    <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              username="dbUser" password="password" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/testdb"/>

</Context>
```

Figure 36: add the database connection in /Tomcat9.0/conf/context.xml

## 6.3 Nutshell Example 5 – Insert Data

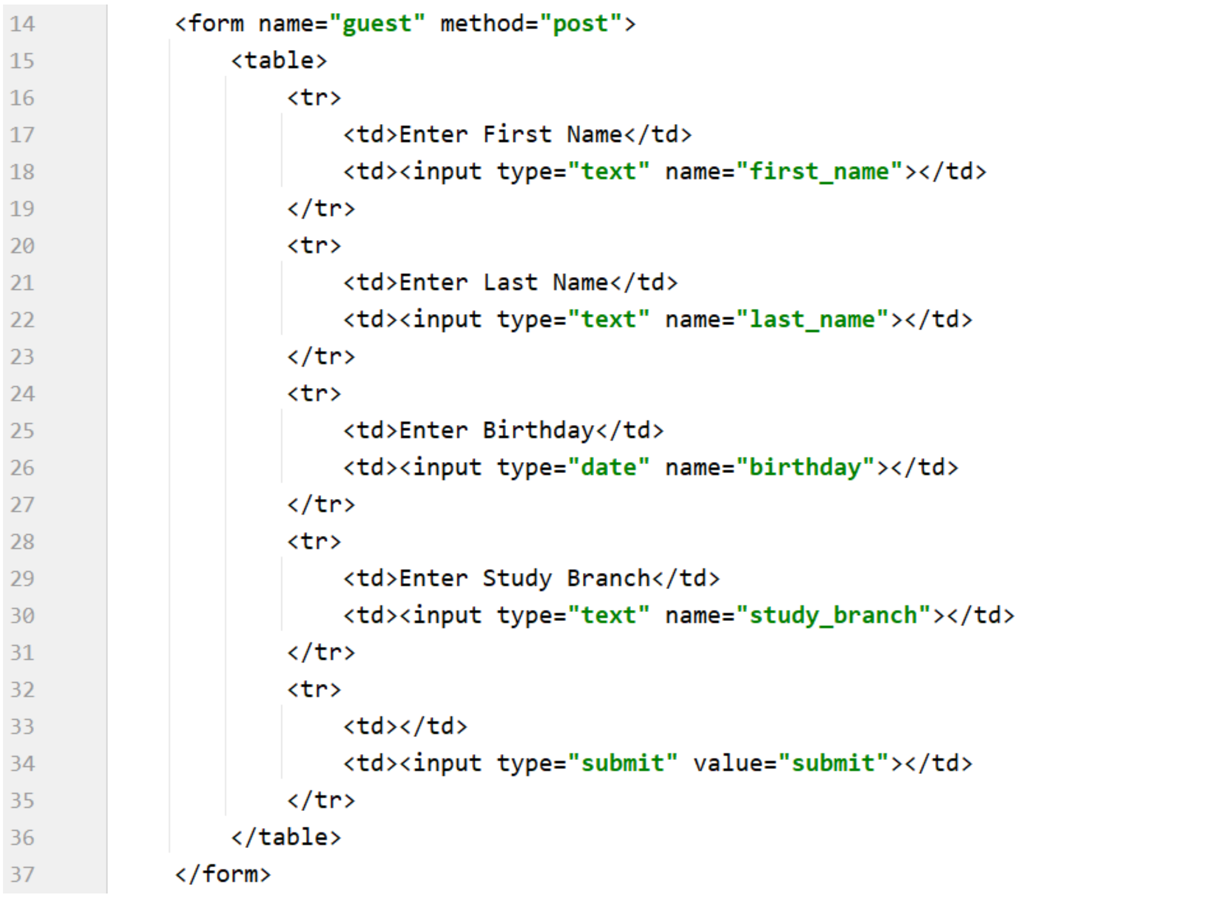
The fifth nutshell example is the first that uses the JSTL resources to access the previously set up database. The example illustrates, how to collect user data within a web application and save it into a database. This is a very common use case which can be implemented with little effort. The values are gathered with an <form> element, similar than in nutshell example 3. By submitting the form, a post method is executed and the data is added to the students table with an INSERT statement.



Nutshell Example 5 - Add a new Student

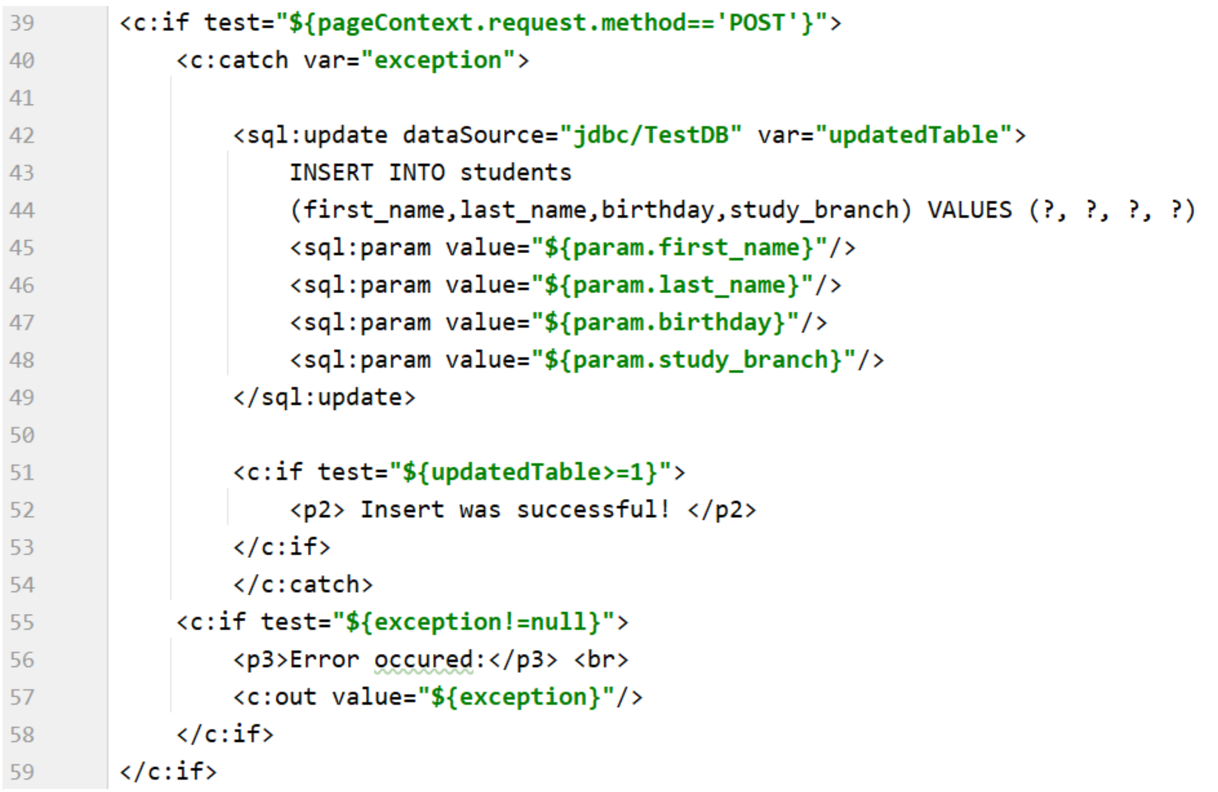
| | |
|---|---|
| Enter First Name | Max |
| Enter Last Name | Mustermann |
| Enter Birthday | 08.07.1997 |
| Enter Study Branch | WINF |
| | submit |

Figure 37: /localhost/Dessert/Nutshell5-InsertData.jsp



```
<form name="guest" method="post">
    <table>
        <tr>
            <td>Enter First Name</td>
            <td><input type="text" name="first_name"></td>
        </tr>
        <tr>
            <td>Enter Last Name</td>
            <td><input type="text" name="last_name"></td>
        </tr>
        <tr>
            <td>Enter Birthday</td>
            <td><input type="date" name="birthday"></td>
        </tr>
        <tr>
            <td>Enter Study Branch</td>
            <td><input type="text" name="study_branch"></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="submit"></td>
        </tr>
    </table>
</form>
```

Figure 38: Nutshell5-InsertData.jsp - <form> element



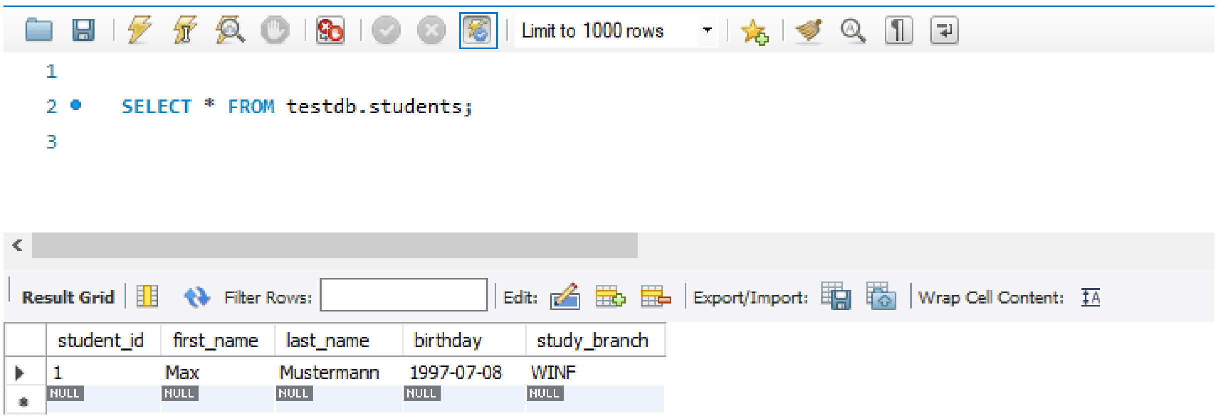
```
<c:if test="${pageContext.request.method=='POST'}">
    <c:catch var="exception">

        <sql:update dataSource="jdbc/TestDB" var="updatedTable">
            INSERT INTO students
            (first_name,last_name,birthday,study_branch) VALUES (?, ?, ?, ?)
            <sql:param value="${param.first_name}"/>
            <sql:param value="${param.last_name}"/>
            <sql:param value="${param.birthday}"/>
            <sql:param value="${param.study_branch}"/>
        </sql:update>

        <c:if test="${updatedTable>=1}">
            <p2> Insert was successful! </p2>
        </c:if>
        </c:catch>
    <c:if test="${exception!=null}">
        <p3>Error occured:</p3> <br>
        <c:out value="${exception}"/>
    </c:if>
</c:if>
```

Figure 39: Nutshell5-InsertData.jsp – <core> and <sql> tag of the JSTL library

Figure 39 shows the script that is executed by the post method.

The <core> library offers a various set of functions. Here the <sql> tag is used for the IF statements and the exception handling is done with the <core> tag. If an error occurs during the execution, it will be caught by the catch block and the exception message would be printed on the web page.

The SQL statement is created and executed in the <sql> tag. By defining the dataSource the path to the database is set. We use the "jdbc/TestDB" variable that we previously added to the context.xml. The syntax of the query is pretty like the regular INSERT statement. The only difference is that we use "?" as placeholders and then set the values by retrieving the input values from the <form>. If the input value types are correct and no other error occurs, the student is successfully added to the database table.

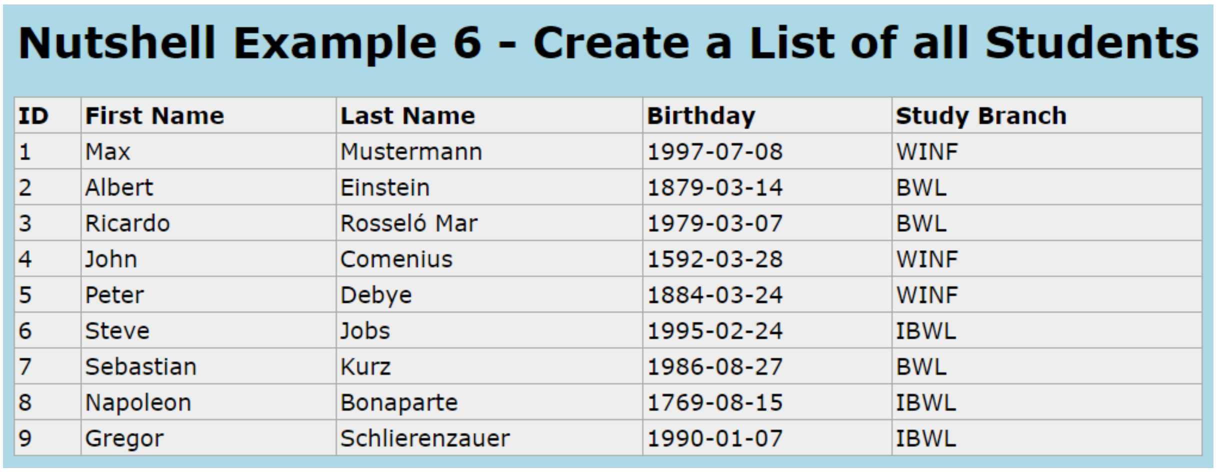


```
SELECT * FROM testdb.students;
```

| student_id | first_name | last_name | birthday | study_branch |
|---|---|---|---|---|
| 1 | Max | Mustermann | 1997-07-08 | WINF |
| NULL | NULL | NULL | NULL | NULL |

Figure 40: MySQL Workbench – select * from students

## 6.4 Nutshell Example 6 – Select Data

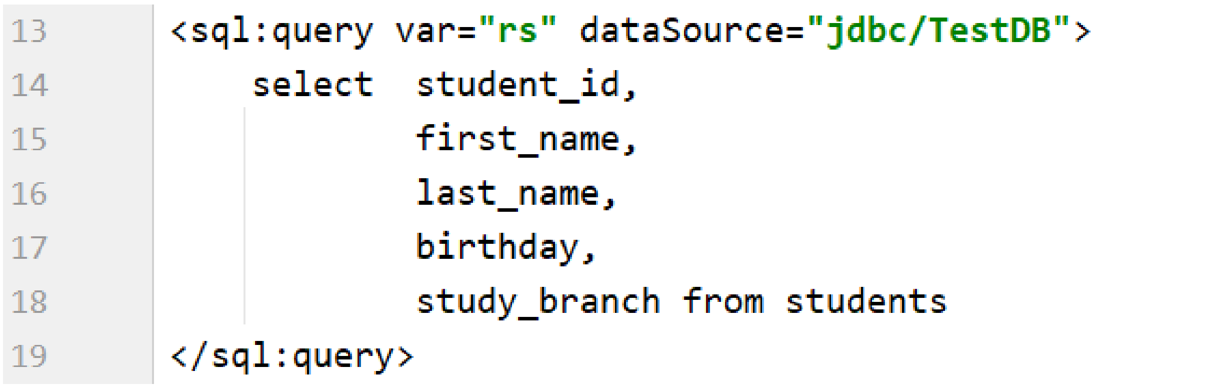


Nutshell Example 6 - Create a List of all Students

| ID | First Name | Last Name | Birthday | Study Branch |
|---|---|---|---|---|
| 1 | Max | Mustermann | 1997-07-08 | WINF |
| 2 | Albert | Einstein | 1879-03-14 | BWL |
| 3 | Ricardo | Rosseló Mar | 1979-03-07 | BWL |
| 4 | John | Comenius | 1592-03-28 | WINF |
| 5 | Peter | Debye | 1884-03-24 | WINF |
| 6 | Steve | Jobs | 1995-02-24 | IBWL |
| 7 | Sebastian | Kurz | 1986-08-27 | BWL |
| 8 | Napoleon | Bonaparte | 1769-08-15 | IBWL |
| 9 | Gregor | Schlierenzauer | 1990-01-07 | IBWL |

Figure 41: localhost/Dessert/Nutshell6-SelectData.jsp

This sixth nutshell example will illustrate, how to retrieve data from the database and present it on a web page. In the previous example we used the INSERT statement to add one student to the database table students. For demonstration purposes, further students were added with the same method. Now we use the <sql> tag for the JSTL library to create a SELECT query. Like before we set the dataSource and additional create the variable "rs" to catch the return values.



```
<sql:query var="rs" dataSource="jdbc/TestDB">
    select  student_id,
            first_name,
            last_name,
            birthday,
            study_branch from students
</sql:query>
```

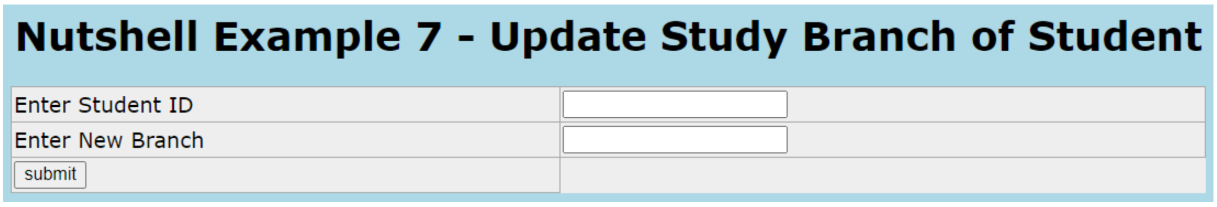
Figure 42: Nutshell6-InsertData.jsp – select statement

To display the values, we use the forEach method of the <core> tag and iterate through the return values. Each row is a student with id, first name, last name, birthday and study branch. First the header of the table columns is set and then a row is added for each student. The "row" variable is accessible during the loop and always points on the current student. The result is a list of all students in the student database table, which is presented in a <table> element.

```
<table>
    <tr>
        <th>ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Birthday</th>
        <th>Study Branch</th>
    </tr>
    <c:forEach var="row" items="${rs.rows}">
        <tr>
            <td>
                <c:out value="${row.student_id}"/>
            </td>
            <td>
                <c:out value="${row.first_name}"/>
            </td>
            <td>
                <c:out value="${row.last_name}"/>
            </td>
            <td>
                <c:out value="${row.birthday}"/>
            </td>
            <td>
                <c:out value="${row.study_branch}"/>
            </td>
        </tr>
    </c:forEach>
</table>
```

Figure 43: Nutshell5-InsertData.jsp – creation of student list

## 6.5 Nutshell Example 7 – Update Data



Nutshell Example 7 - Update Study Branch of Student

| | |
|---|---|
| Enter Student ID | |
| Enter New Branch | |
| submit | |

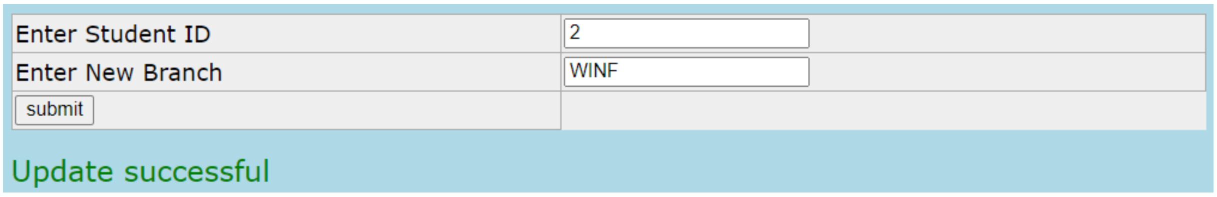
Figure 44: localhost/Dessert/Nutshell7-UpdateData

The seventh nutshell example is about updating values in the database tables. This will be demonstrated by changing the study branch of a single student. To update with SQL two things are required. First, the new value of the attribute to update and second, criteria to clearly determine which entries should be updated. Students normally have a unique student id. In contrast to the name or other variables, there cannot be duplicate values for the id. Therefore, it is perfect to identify one single student.

```
        <sql:update dataSource="jdbc/TestDB" var="updatedTable">
            UPDATE students SET study_branch=? WHERE student_id = ?;
            <sql:param value="${param.study_branch}"/>
            <sql:param value="${param.student_id}"/>
        </sql:update>
```

Figure 45: Nutshell7-UpdateData.jsp

By using the <sql> tag, an UPDATE statement can be created. The creation is similar to the fifth nutshell example, where an INSERT was used. The values are collected with the <form> element and a submit button executes the script.

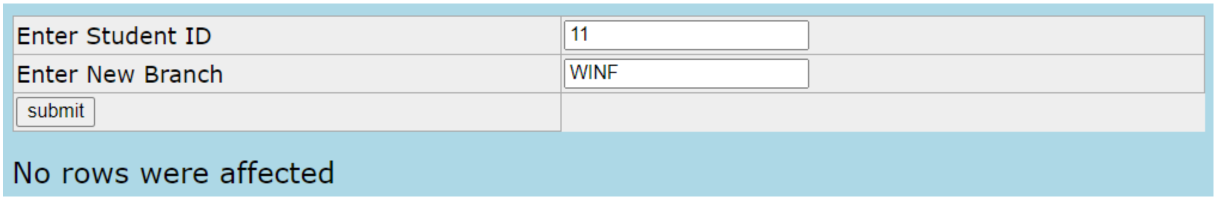


| | |
|---|---|
| Enter Student ID | 2 |
| Enter New Branch | WINF |
| submit | |

Update successful

Figure 46: localhost/Dessert/Nutshell7-UpdateData.jsp - update successful

If no error occurred and a row is updated, the message "Update successful" will be printed. A transaction without any changes on the contrary, will result in a different message. The user will be informed by the message "No rows were affected", because no student with this id exists and thereby no update happened.



| | |
|---|---|
| Enter Student ID | 11 |
| Enter New Branch | WINF |
| submit | |

No rows were affected

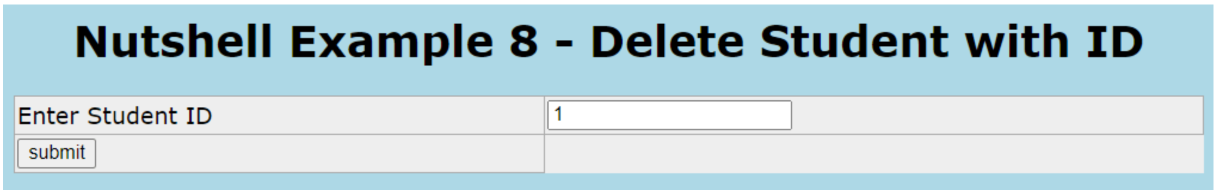
Figure 47: localhost/Dessert/Nutshell7-UpdateData.jsp – no rows updated

To handle the different return results, it is recommended to use if statements. Each return possibility will be treated different. In the worst case, there will be notification in form of an error text with the exception message appended.

```
        <c:if test="${updatedTable>=1}">
            <p2>Update successful</p2>
        </c:if>

        <c:if test="${updatedTable<=0}">
            <p1>No rows were affected</p1>
        </c:if>

        <c:if test="${exception!=null}">
            <p3>Error occured:</p3> <br>
            <c:out value="${exception}"/>
        </c:if>
```

Figure 48: Nutshell7-UpdateData - exeption handling

## 6.6 Nutshell Example 8 – Delete Data



Nutshell Example 8 - Delete Student with ID

| | |
|---|---|
| Enter Student ID | 1 |

Figure 49: localhost/Dessert/Nutshell8-DeleteData.jsp

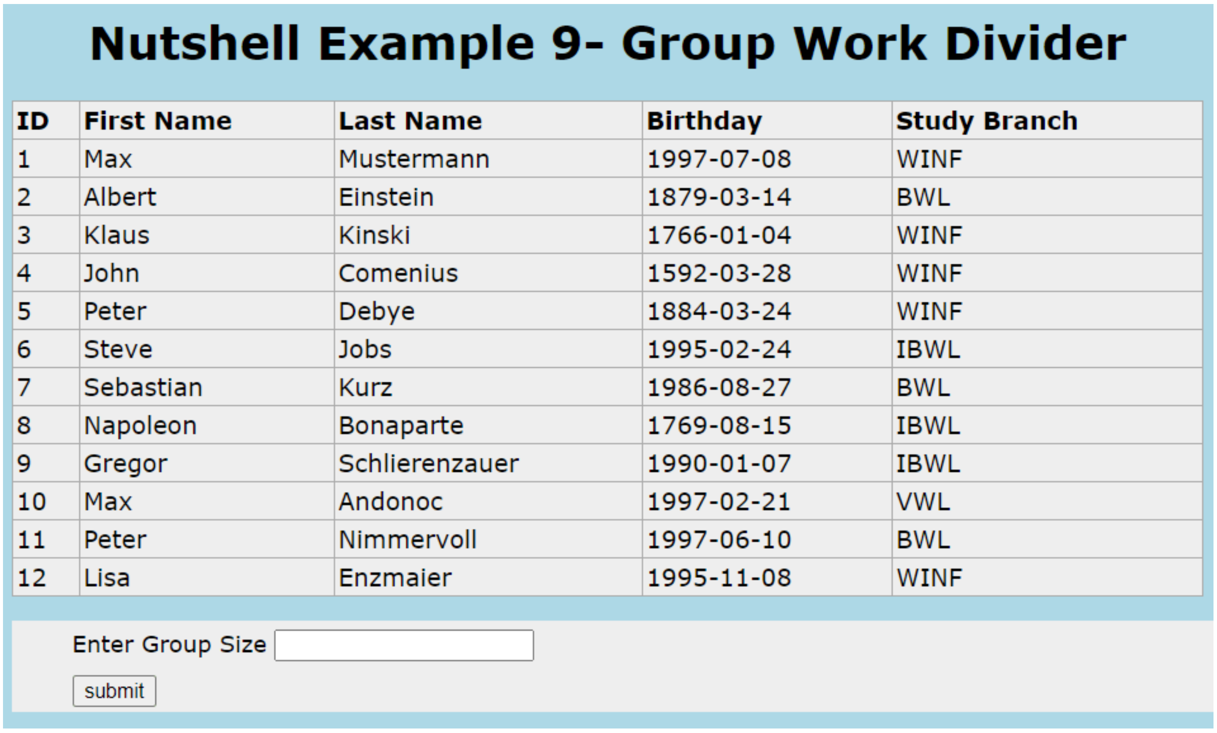
The eighth nutshell example demonstrates the deletion of a student from the database. Similar to the update, the deletion requires criteria to determine the entries to delete. Again, the student id is used, because of its uniqueness. If the transaction is successful, the student is deleted from the table and only a ROLLBACK of the database could recover the data.

```
<sql:update dataSource="jdbc/TestDB" var="updatedTable">
    DELETE FROM students WHERE student_id = ?;
    <sql:param value="${param.student_id}"/>
</sql:update>
```

Figure 50: Nutshell8-DeleteData.jsp

The script from **Figure 48** can also be implemented in this nutshell example to improve usability of the web application, by informing the user about changes.

## 6.7 Nutshell Example 9 – Group Work Organizer



Nutshell Example 9- Group Work Divider

| ID | First Name | Last Name | Birthday | Study Branch |
|---|---|---|---|---|
| 1 | Max | Mustermann | 1997-07-08 | WINF |
| 2 | Albert | Einstein | 1879-03-14 | BWL |
| 3 | Klaus | Kinski | 1766-01-04 | WINF |
| 4 | John | Comenius | 1592-03-28 | WINF |
| 5 | Peter | Debye | 1884-03-24 | WINF |
| 6 | Steve | Jobs | 1995-02-24 | IBWL |
| 7 | Sebastian | Kurz | 1986-08-27 | BWL |
| 8 | Napoleon | Bonaparte | 1769-08-15 | IBWL |
| 9 | Gregor | Schlierenzauer | 1990-01-07 | IBWL |
| 10 | Max | Andonoc | 1997-02-21 | VWL |
| 11 | Peter | Nimmervoll | 1997-06-10 | BWL |
| 12 | Lisa | Enzmaier | 1995-11-08 | WINF |

Figure 51: localhost/Dessert/Nutshell9-GroupWorkOrganizer.jsp

The ninth and last nutshell presents an overview of all students. Additionally, the example offers the functionality to divide the students into separate groups. The group size is defined by an input field and with a submit button the function can be executed.

The students table, which presents the students data, is loaded with the SELECT query like in nutshell example 6. The function to divide the students in different groups happens in the oRexx servlet. By clicking the submit button of the <form> element a POST method gets executed with the input value for the group size added into the header of the request.

```
    <form action="${myURL}" method="post"/>
            <ul>
                <li>
                <td>Enter Group Size</td>
                    <td><input type="text" name="input"></td>
                </li>
                <li>
                    <input type="submit" value="submit">
                </li>
            </ul>
    <form/>
```

Figure 52: Nutshell9-GroupWorkDivider.jsp

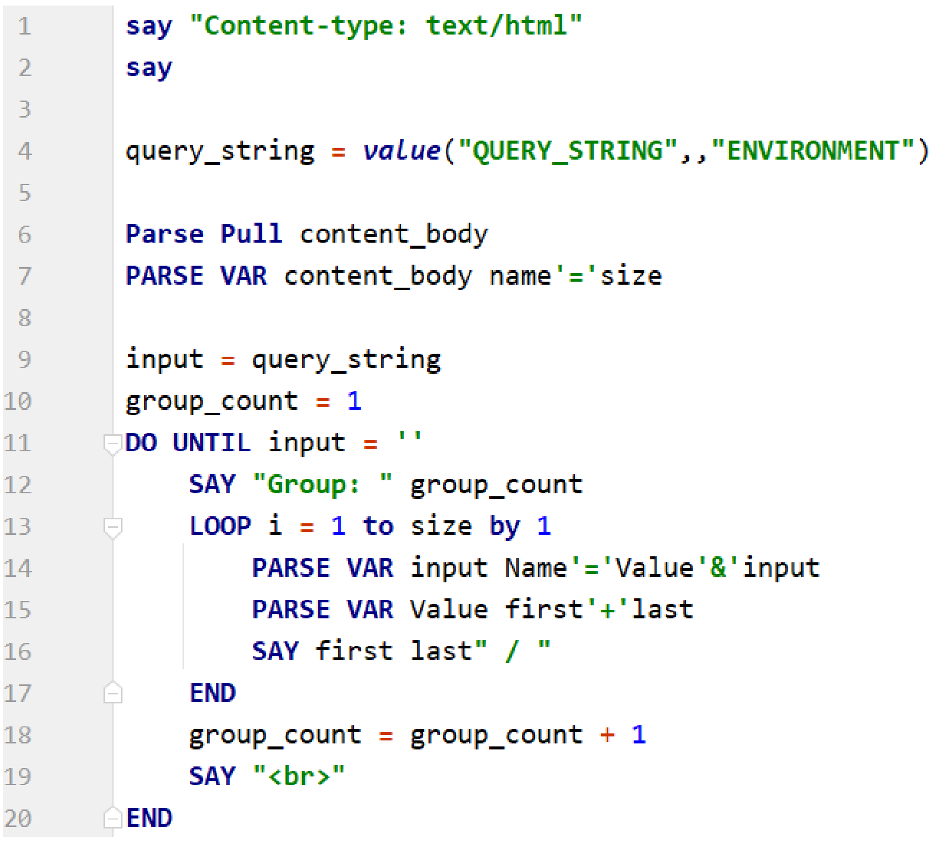
To divide the students into teams, it is also necessary to have access to their names. Therefore, some of the values returned by the SELECT statement must be added to the request before executing it. This can be done by using the URL function of the <core> tag. The value of the URL is the oRexx servlet. Next, iterate through the student data and add every name parameter to this URL. By doing that, the values will be added to the body of the request, as with the GET method. Last, set the action of the <form> with this URL, so if the <form> is submitted a POST request with the student names in the body is sent.[11]

```
<c:url value="cgi-bin/groupStudents.rexx" var="myURL">
    <c:forEach var="row" items="${rs.rows}">
        <c:param name="name" value="${row.first_name} ${row.last_name}"/>
    </c:forEach>
</c:url>
```

Figure 53: Nutshell9-GroupWorkDivider.jsp - add students to request body

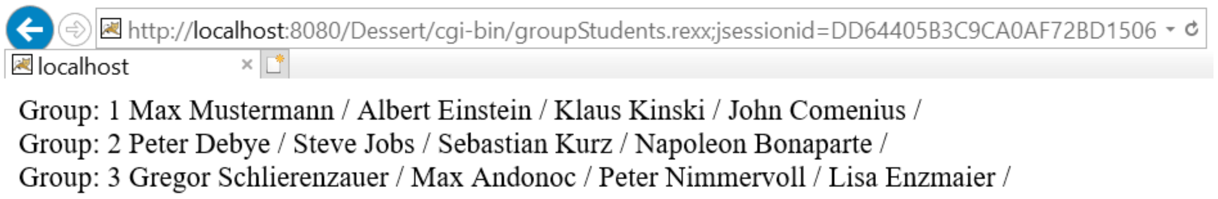
Figure 54 shows the complete groupStudents.rexx servlet. To implement the oRexx servlet, it is necessary to follow the same steps mentioned for nutshell example 3 and 4.

In the servlet the input values, both header and body, are retrieved first. To divide the students an encapsulated loop is used, that iterates through all name values from the request body. The inner loop adds students to a group till its counter reaches the value of the size variable. Then the counter is reset to 1 and a new Group is created. In this example, the groups are only printed to the web page and not further processed. The members are separated by a "/" and between each group a <br> tag is added to force a new text line.



```
say "Content-type: text/html"
say

query_string = value("QUERY_STRING",,"ENVIRONMENT")

Parse Pull content_body
PARSE VAR content_body name'='size

input = query_string
group_count = 1
DO UNTIL input = ''
    SAY "Group: " group_count
    LOOP i = 1 to size by 1
        PARSE VAR input Name'='Value'&'input
        PARSE VAR Value first'+'last
        SAY first last" / "
    END
    group_count = group_count + 1
    SAY "<br>"
END
```

Figure 54: /WEB-INF/cgi/groupStudents.rexx



http://localhost:8080/Dessert/cgi-bin/groupStudents.rexx;jsessionid=DD64405B3C9CA0AF72BD1506

localhost

```
Group: 1 Max Mustermann / Albert Einstein / Klaus Kinski / John Comenius /
Group: 2 Peter Debye / Steve Jobs / Sebastian Kurz / Napoleon Bonaparte /
Group: 3 Gregor Schlierenzauer / Max Andonoc / Peter Nimmervoll / Lisa Enzmaier /
```

Figure 55: /localhost/Dessert/cgi-bin/groupStudents.rexx – output of servlet

# 7 Conclusion

## 7.1 Summary

To summarize, the work with Apache Tomcat and the development of the nutshell examples was a really challenging but also satisfying project. In contrast to regular research papers, creating a cookbook with instructions and nutshell examples is much more practice oriented and animates to get creative.

After learning the basics of oRexx and the associated tag libraries the development becomes very intuitive. This is mainly due to the simple syntax of the oRexx language. In combination with JSP it is no challenge to create fancy web pages that meet the requirements for small use cases. Therefore, it is a conceivable option for small or medium businesses, which are interested in optimizing or automating processes.

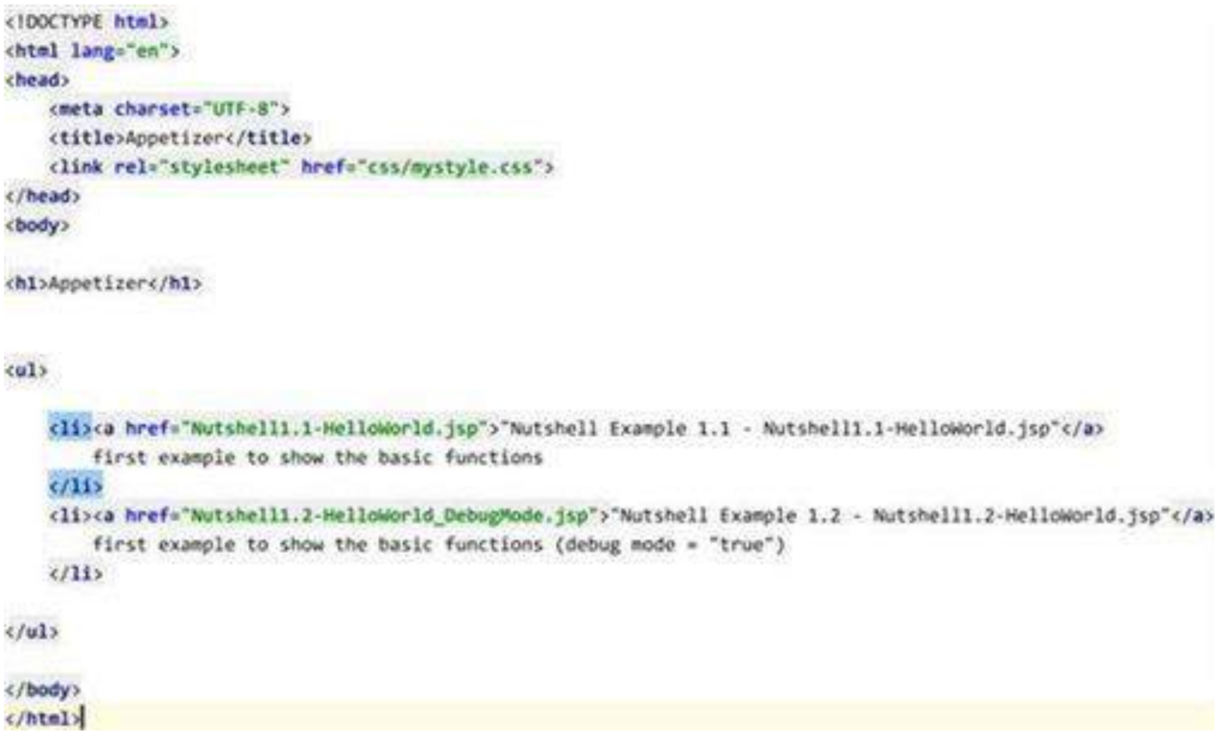
## 7.2 Outlook

The last nutshell example demonstrates a combination of nearly all features mentioned in this paper. Of course, this is not the limit of this resources. There are many improvements, which were not implemented due to time constraints. The BSF4ooRexx package enables the use of every JAVA class and thereby offers nearly unlimited features. Another aspect is the database. The database used in the nutshell example contained only a single table. A more complex database structure and SELECT queries with JOINs over multiple tables would satisfy more complex use cases. Further, it is possible to use different kinds of databases, like SQLite.

To wrap it all up, the covered topics offer a solid basis for the development of web applications with oRexx. I can personally recommend inexperienced students with an interest in web development to start with Apache Tomcat and oRexx.

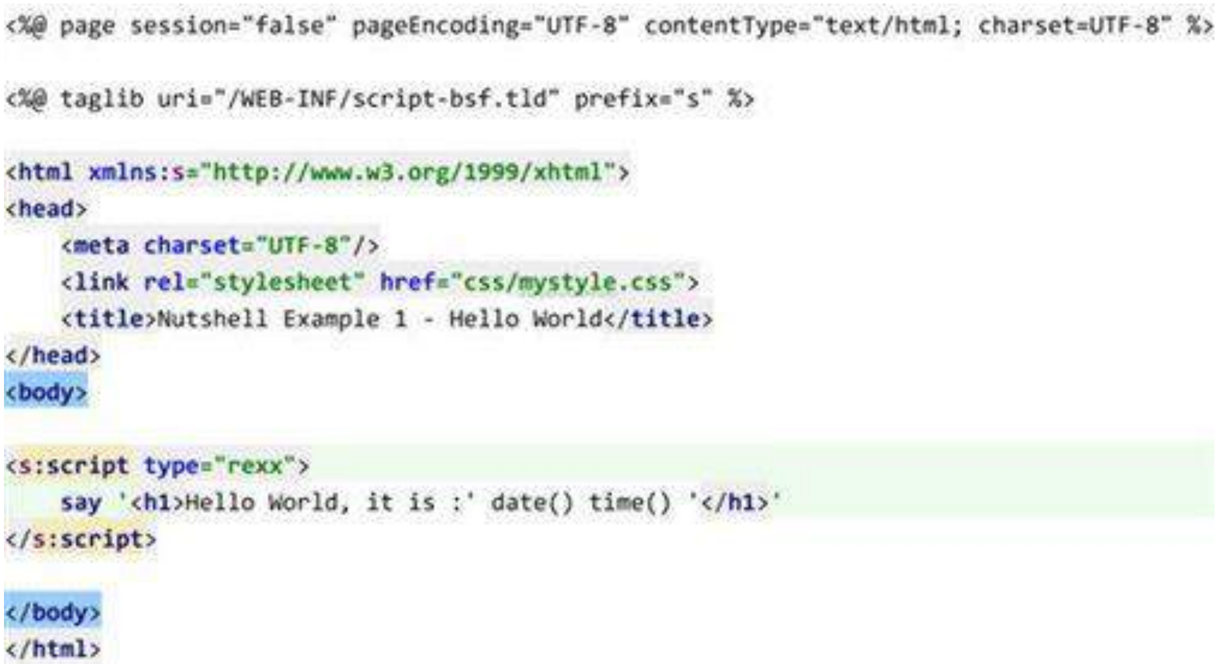
## Appendix

### Appetizer – index.html



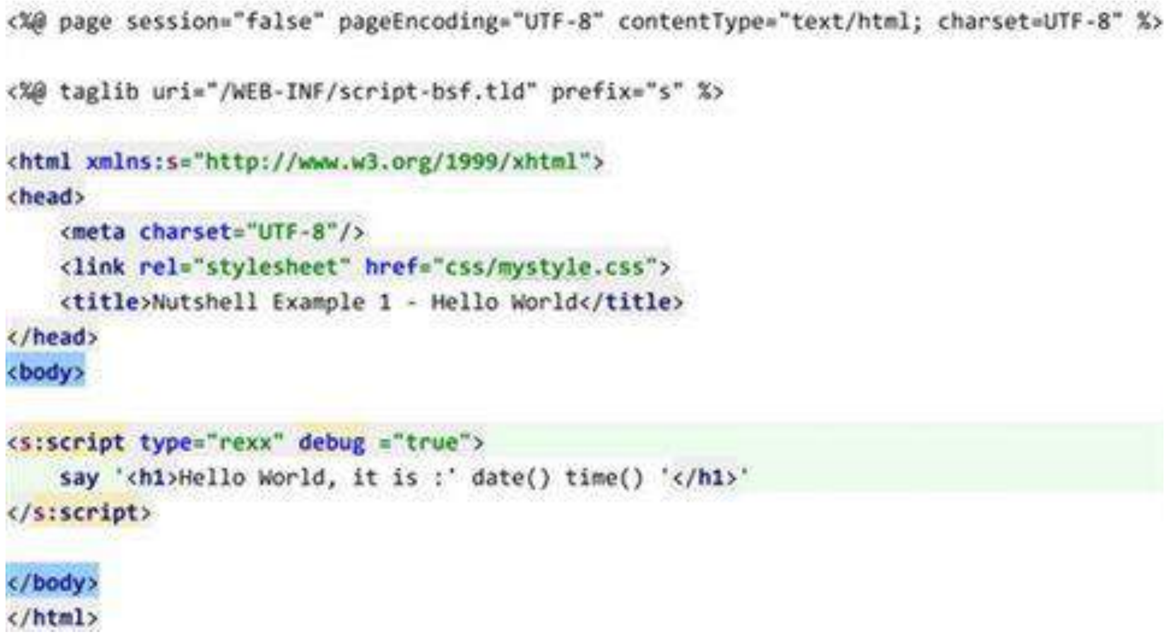
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Appetizer</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>
<body>

<h1>Appetizer</h1>


<ul>

    <li><a href="Nutshell1.1-HelloWorld.jsp">"Nutshell Example 1.1 - Nutshell1.1-HelloWorld.jsp"</a>
        first example to show the basic functions
    </li>
    <li><a href="Nutshell1.2-HelloWorld_DebugMode.jsp">"Nutshell Example 1.2 - Nutshell1.2-HelloWorld.jsp"</a>
        first example to show the basic functions (debug mode = "true")
    </li>

</ul>

</body>
</html>
```

### Nutshell1.1-HelloWorld.jsp



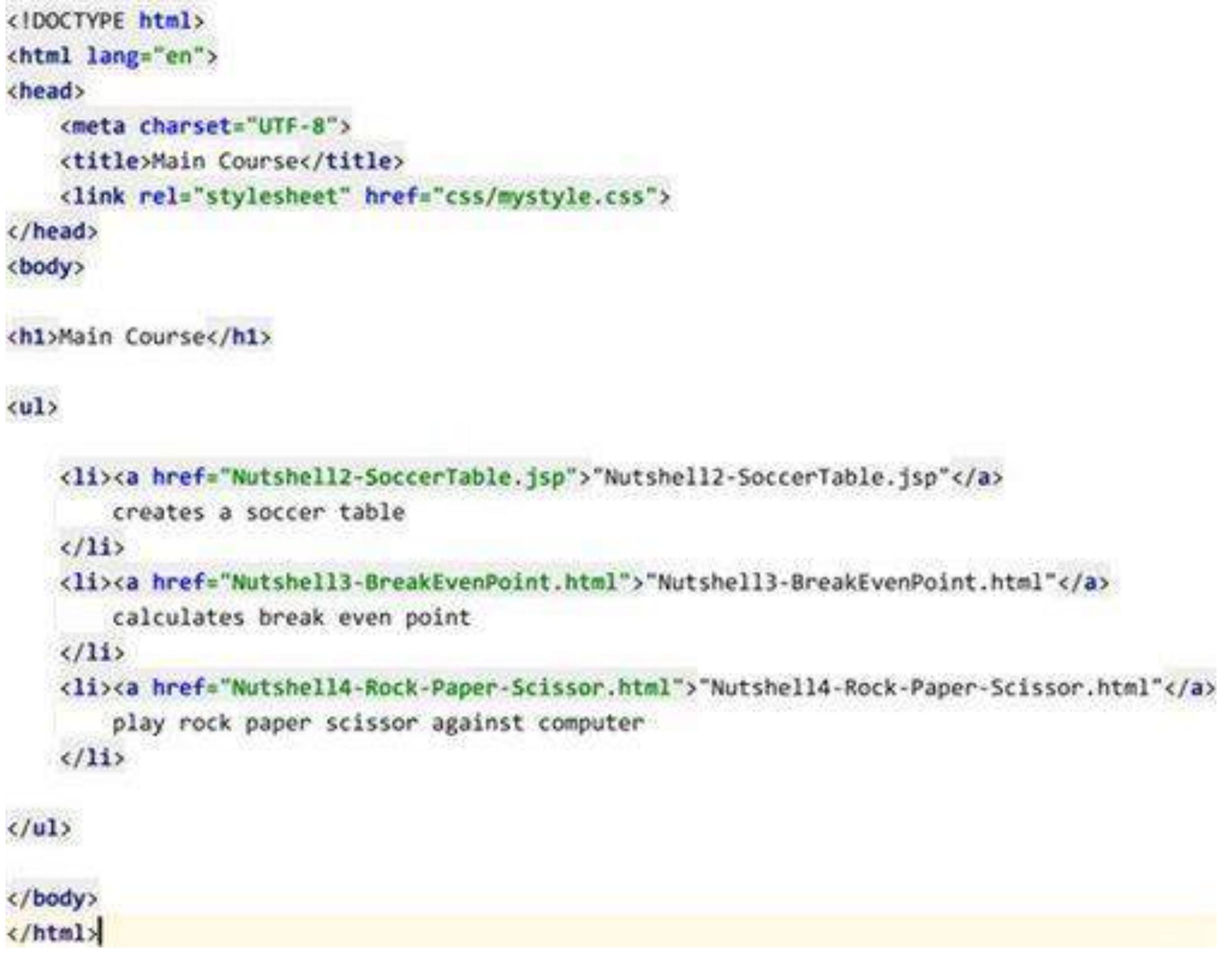
```jsp
<%@ page session="false" pageEncoding="UTF-8" contentType="text/html; charset=UTF-8" %>

<%@ taglib uri="/WEB-INF/script-bsf.tld" prefix="s" %>

<html xmlns:s="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8"/>
    <link rel="stylesheet" href="css/mystyle.css">
    <title>Nutshell Example 1 - Hello World</title>
</head>
<body>

<s:script type="rexx">
    say '<h1>Hello World, it is :' date() time() '</h1>'
</s:script>

</body>
</html>
```

## Nutshell1.2-HelloWorld_DebugMode.jsp



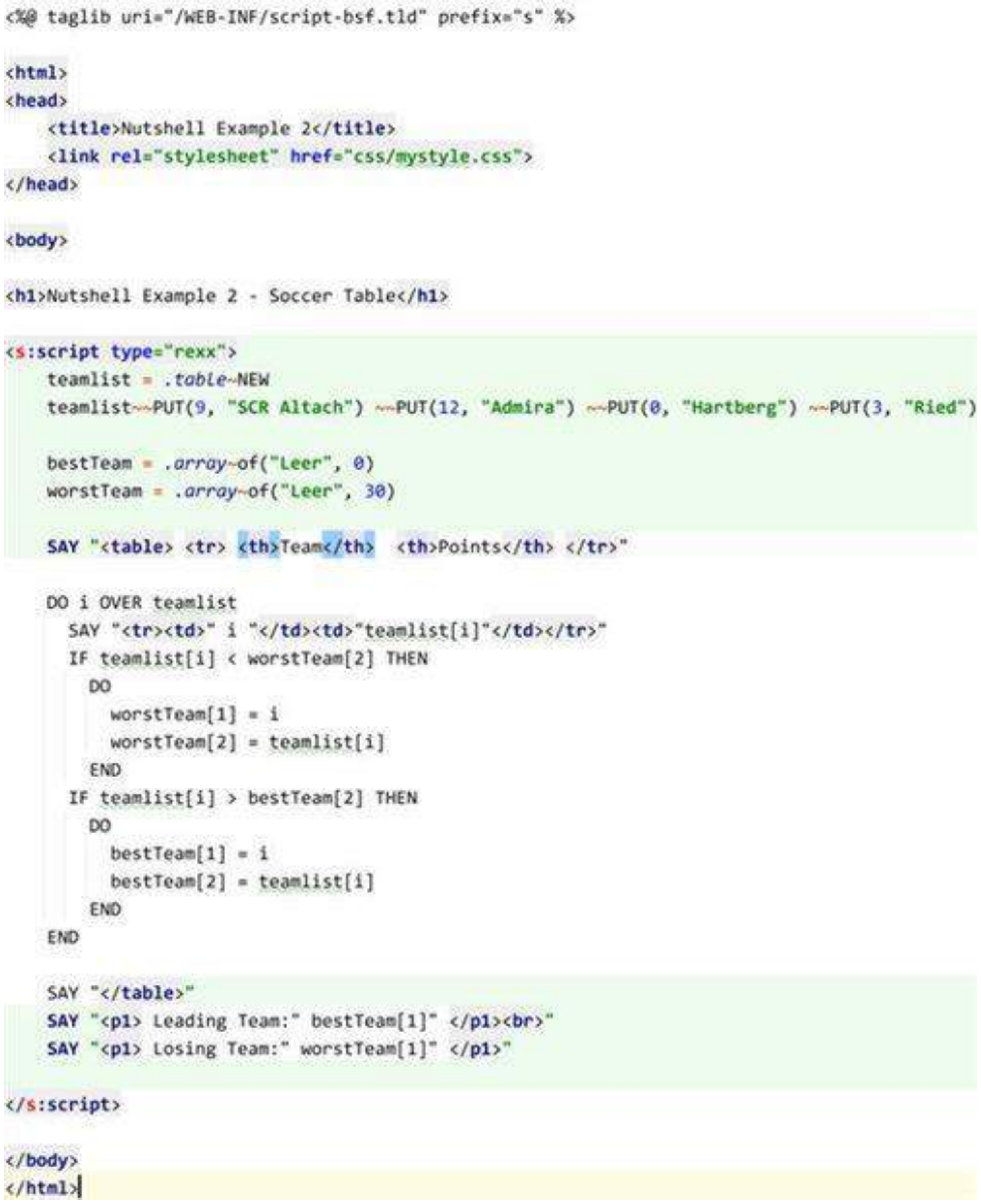
```
<%@ page session="false" pageEncoding="UTF-8" contentType="text/html; charset=UTF-8" %>

<%@ taglib uri="/WEB-INF/script-bsf.tld" prefix="s" %>

<html xmlns:s="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8"/>
    <link rel="stylesheet" href="css/mystyle.css">
    <title>Nutshell Example 1 - Hello World</title>
</head>
<body>

<s:script type="rexx" debug ="true">
    say '<h1>Hello World, it is :' date() time() '</h1>'
</s:script>

</body>
</html>
```

## Main Course -index.html



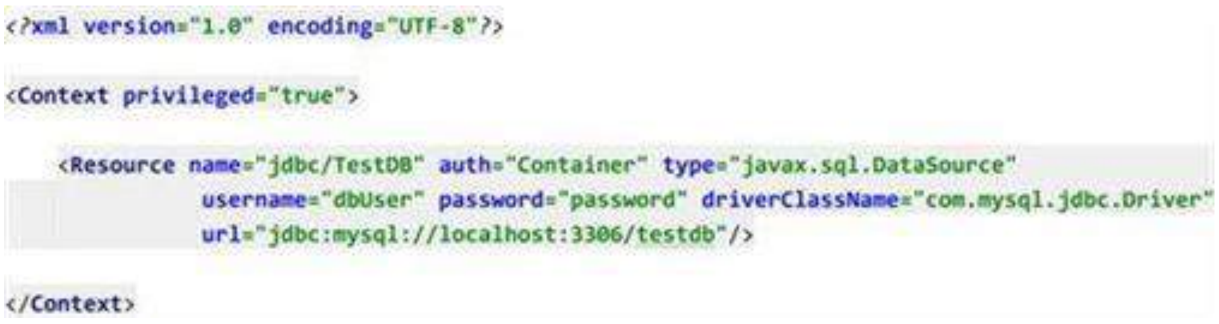
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Main Course</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>
<body>

<h1>Main Course</h1>

<ul>

    <li><a href="Nutshell2-SoccerTable.jsp">"Nutshell2-SoccerTable.jsp"</a>
        creates a soccer table
    </li>
    <li><a href="Nutshell3-BreakEvenPoint.html">"Nutshell3-BreakEvenPoint.html"</a>
        calculates break even point
    </li>
    <li><a href="Nutshell4-Rock-Paper-Scissor.html">"Nutshell4-Rock-Paper-Scissor.html"</a>
        play rock paper scissor against computer
    </li>

</ul>

</body>
</html>
```

## Nutshell2 – SoccerTable.jsp



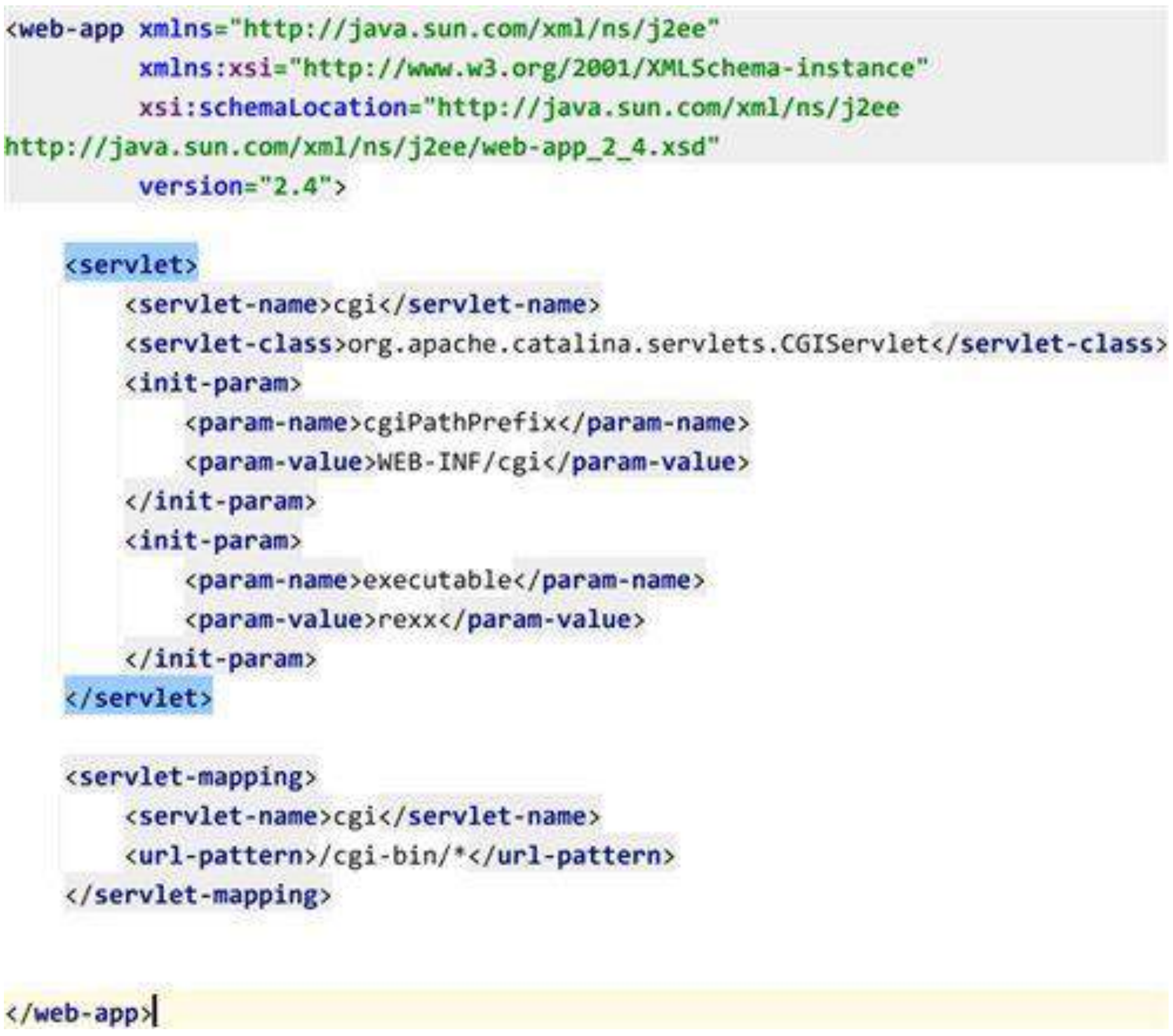
```
<%@ taglib uri="/WEB-INF/script-bsf.tld" prefix="s" %>

<html>
<head>
    <title>Nutshell Example 2</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>

<body>

<h1>Nutshell Example 2 - Soccer Table</h1>

<s:script type="rexx">
    teamlist = .table~NEW
    teamlist~~PUT(9, "SCR Altach") ~~PUT(12, "Admira") ~~PUT(0, "Hartberg") ~~PUT(3, "Ried")

    bestTeam = .array~of("Leer", 0)
    worstTeam = .array~of("Leer", 30)

    SAY "<table> <tr> <th>Team</th>  <th>Points</th> </tr>"

    DO i OVER teamlist
      SAY "<tr><td>" i "</td><td>"teamlist[i]"</td></tr>"
      IF teamlist[i] < worstTeam[2] THEN
        DO
          worstTeam[1] = i
          worstTeam[2] = teamlist[i]
        END
      IF teamlist[i] > bestTeam[2] THEN
        DO
          bestTeam[1] = i
          bestTeam[2] = teamlist[i]
        END
    END

    SAY "</table>"
    SAY "<p1> Leading Team:" bestTeam[1]" </p1><br>"
    SAY "<p1> Losing Team:" worstTeam[1]" </p1>"

</s:script>

</body>
</html>
```

## Tomcat9.0/conf/context.xml



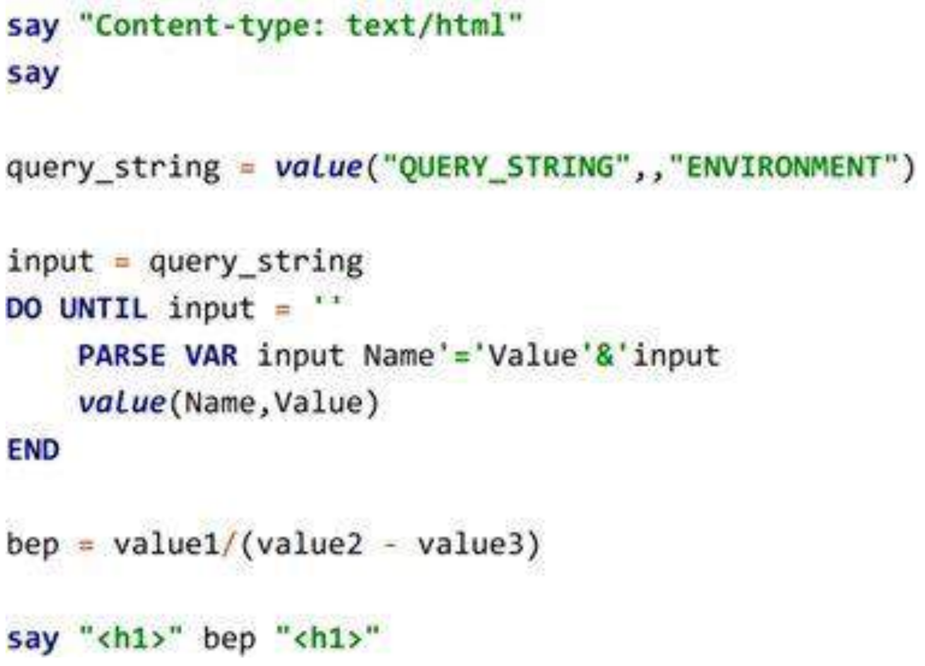
```
<?xml version="1.0" encoding="UTF-8"?>

<Context privileged="true">

    <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              username="dbUser" password="password" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/testdb"/>

</Context>
```

## MainCourse/WEB-INF/web.xml



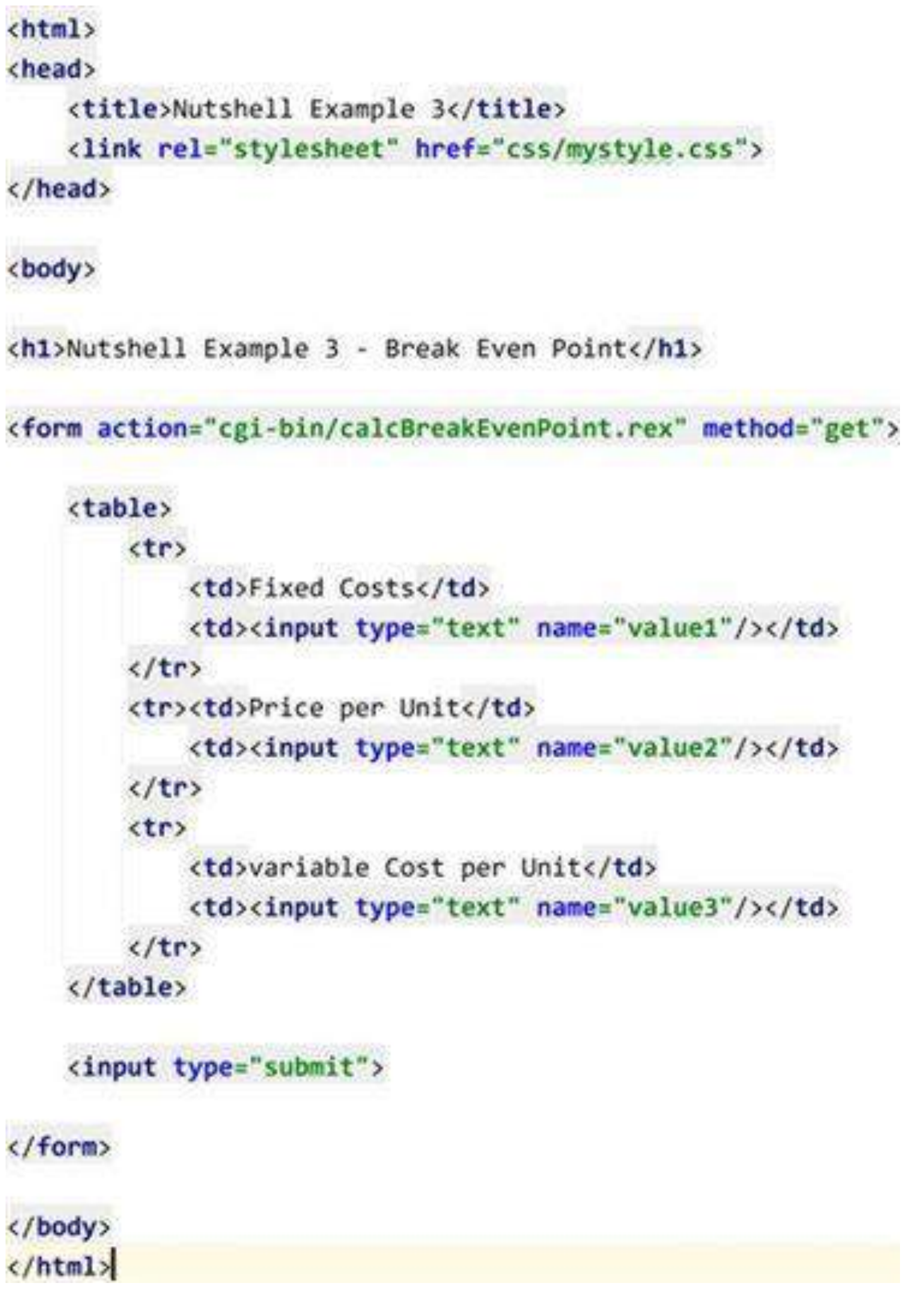
```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">

    <servlet>
        <servlet-name>cgi</servlet-name>
        <servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>
        <init-param>
            <param-name>cgiPathPrefix</param-name>
            <param-value>WEB-INF/cgi</param-value>
        </init-param>
        <init-param>
            <param-name>executable</param-name>
            <param-value>rexx</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>cgi</servlet-name>
        <url-pattern>/cgi-bin/*</url-pattern>
    </servlet-mapping>


</web-app>
```

## MainCourse/WEB-INF/cgi/calcBreakEvenPoint.rex



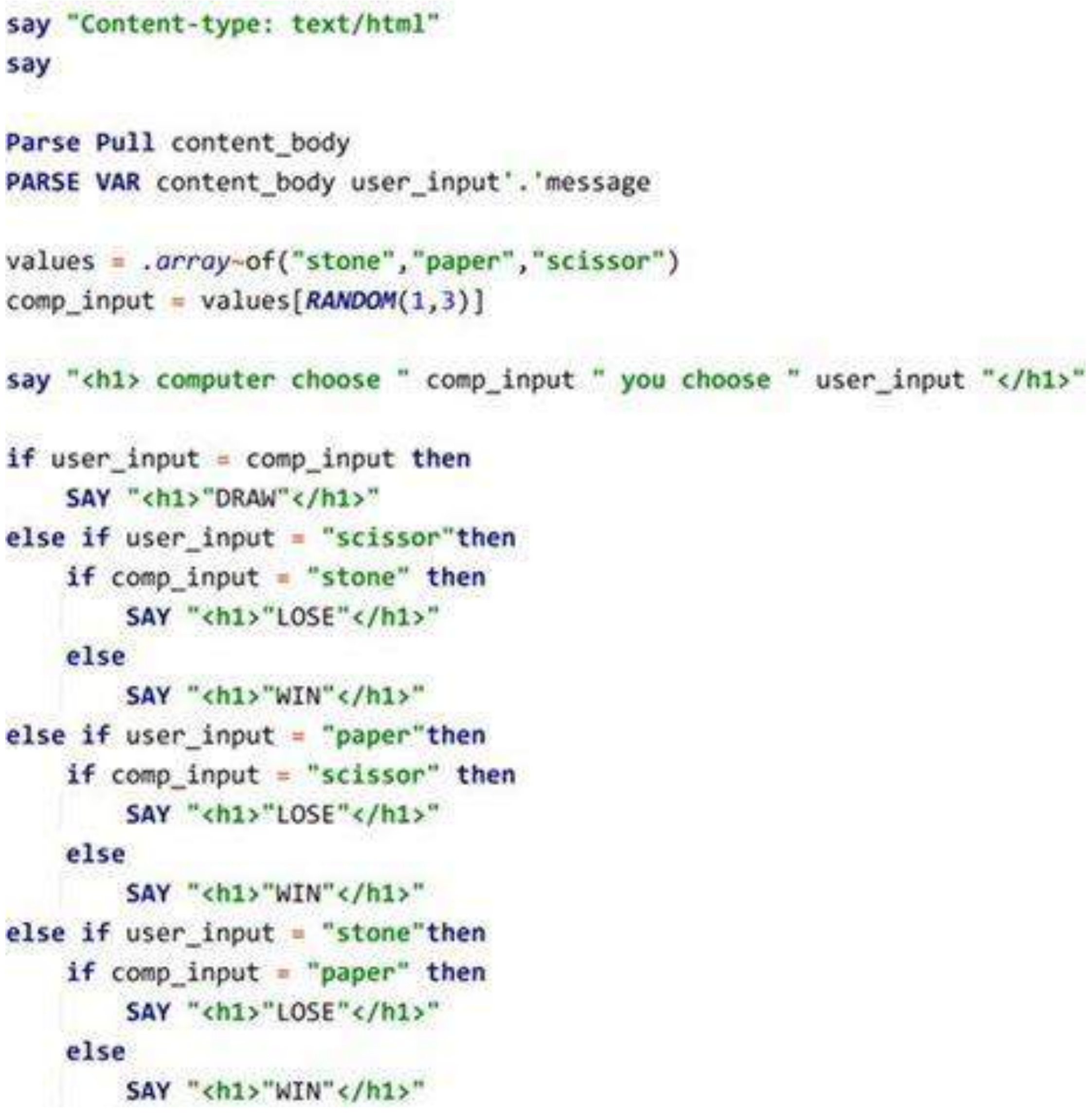
```
say "Content-type: text/html"
say

query_string = value("QUERY_STRING",,"ENVIRONMENT")

input = query_string
DO UNTIL input = ''
    PARSE VAR input Name'='Value'&'input
    value(Name,Value)
END

bep = value1/(value2 - value3)

say "<h1>" bep "<h1>"
```

## Nutshell3 – BreakEvenPoint.html



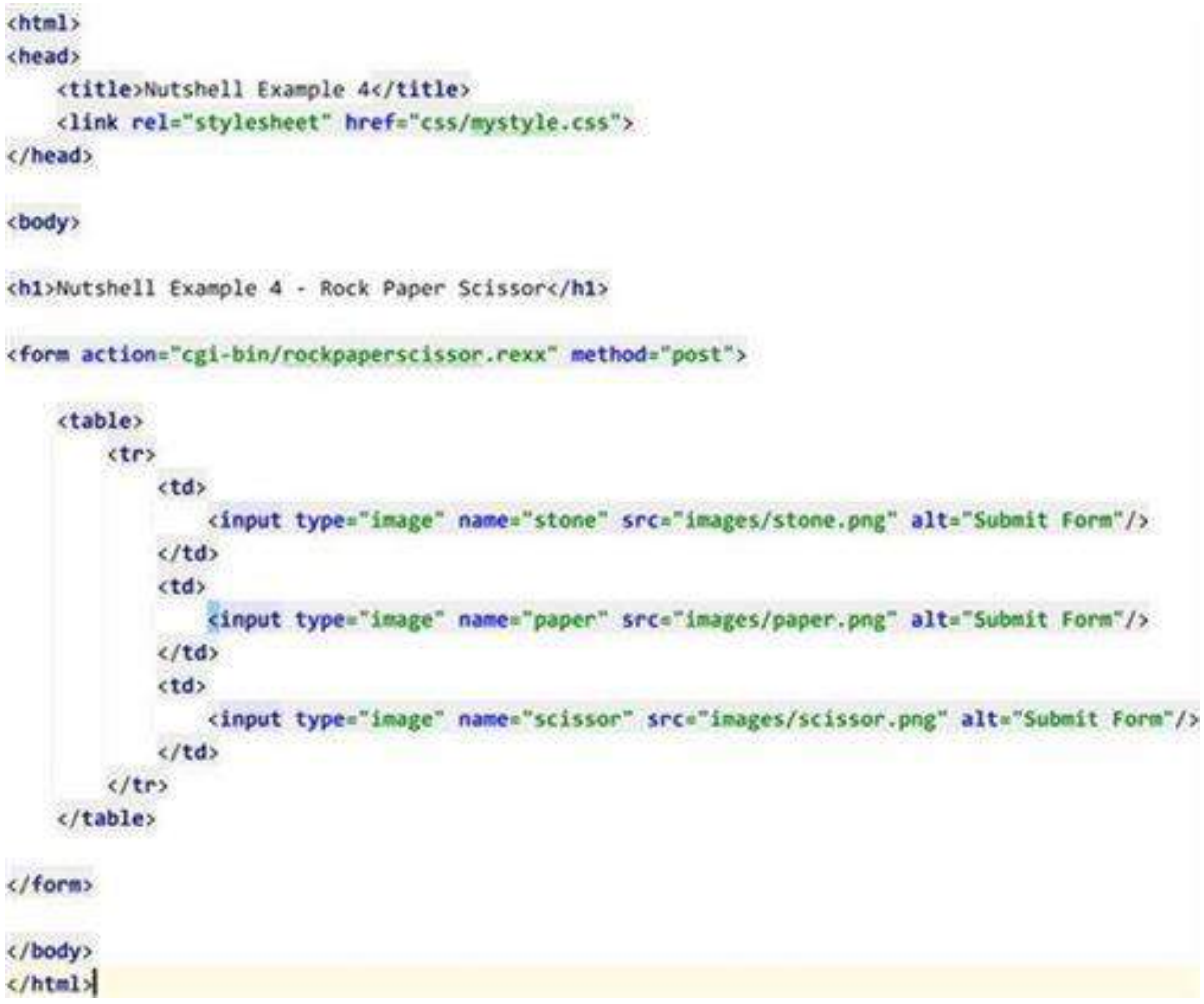
```html
<html>
<head>
    <title>Nutshell Example 3</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>

<body>

<h1>Nutshell Example 3 - Break Even Point</h1>

<form action="cgi-bin/calcBreakEvenPoint.rex" method="get">

    <table>
        <tr>
            <td>Fixed Costs</td>
            <td><input type="text" name="value1"/></td>
        </tr>
        <tr><td>Price per Unit</td>
            <td><input type="text" name="value2"/></td>
        </tr>
        <tr>
            <td>variable Cost per Unit</td>
            <td><input type="text" name="value3"/></td>
        </tr>
    </table>

    <input type="submit">

</form>

</body>
</html>
```

## MainCourse/WEB-INF/cgi/rockpaperscissor.rexx



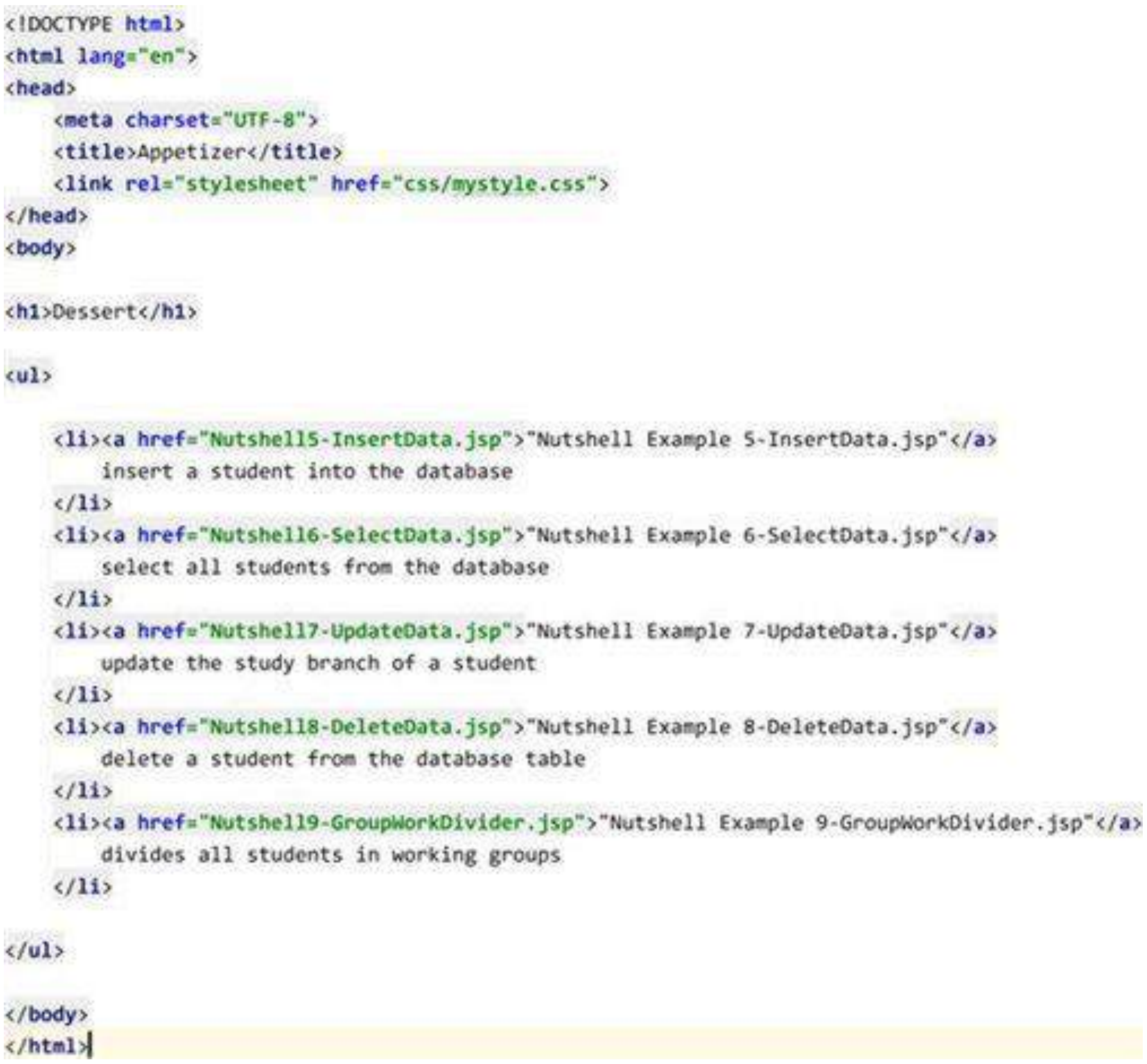
```
say "Content-type: text/html"
say

Parse Pull content_body
PARSE VAR content_body user_input'.'message

values = .array~of("stone","paper","scissor")
comp_input = values[RANDOM(1,3)]

say "<h1> computer choose " comp_input " you choose " user_input "</h1>"

if user_input = comp_input then
    SAY "<h1>"DRAW"</h1>"
else if user_input = "scissor"then
    if comp_input = "stone" then
        SAY "<h1>"LOSE"</h1>"
    else
        SAY "<h1>"WIN"</h1>"
else if user_input = "paper"then
    if comp_input = "scissor" then
        SAY "<h1>"LOSE"</h1>"
    else
        SAY "<h1>"WIN"</h1>"
else if user_input = "stone"then
    if comp_input = "paper" then
        SAY "<h1>"LOSE"</h1>"
    else
        SAY "<h1>"WIN"</h1>"
```

## Nutshell4-Rock-Paper-Scissor.html



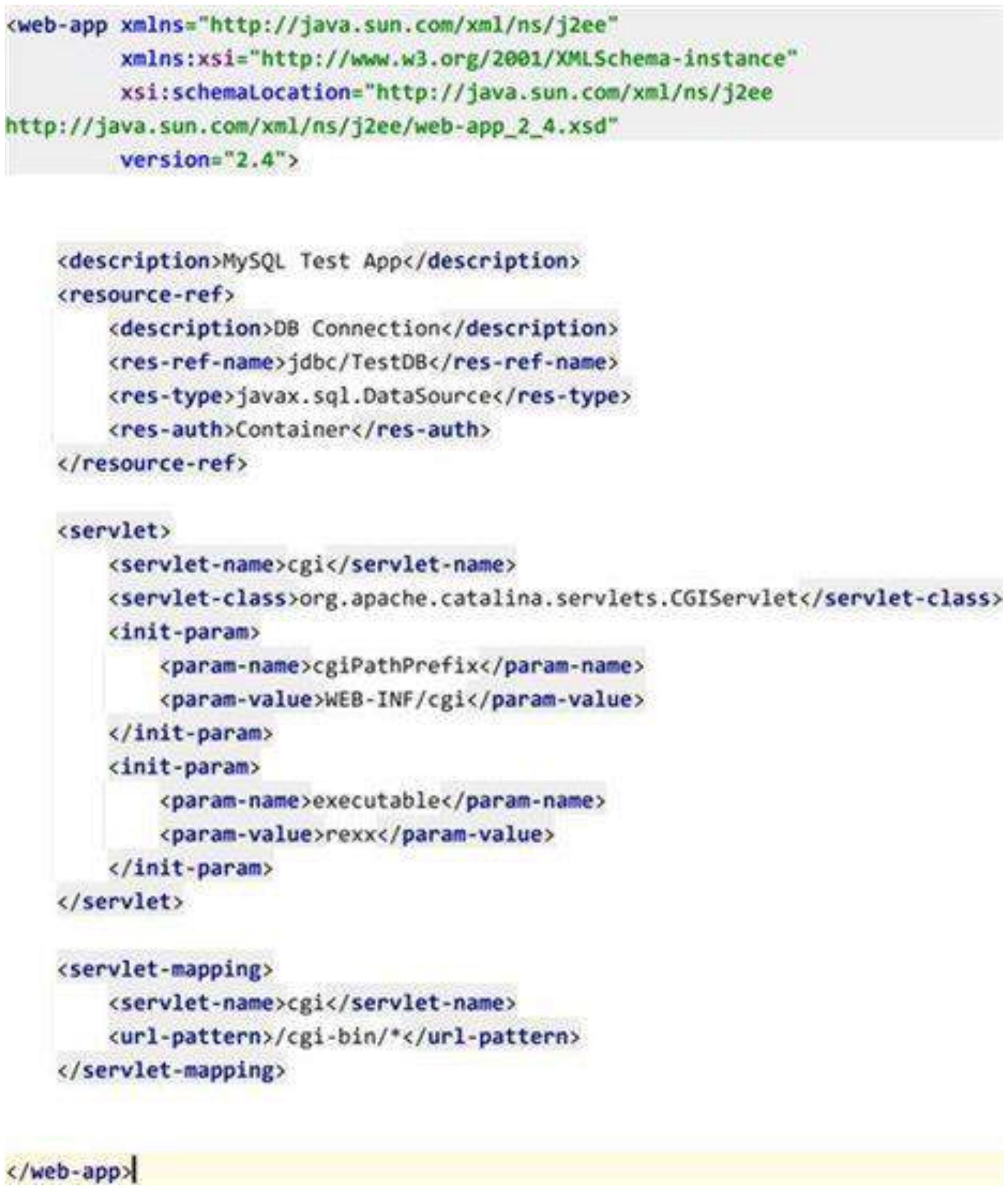
```
<html>
<head>
    <title>Nutshell Example 4</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>

<body>

<h1>Nutshell Example 4 - Rock Paper Scissor</h1>

<form action="cgi-bin/rockpaperscissor.rexx" method="post">

    <table>
        <tr>
            <td>
                <input type="image" name="stone" src="images/stone.png" alt="Submit Form"/>
            </td>
            <td>
                <input type="image" name="paper" src="images/paper.png" alt="Submit Form"/>
            </td>
            <td>
                <input type="image" name="scissor" src="images/scissor.png" alt="Submit Form"/>
            </td>
        </tr>
    </table>

</form>

</body>
</html>
```

## Dessert – index.html



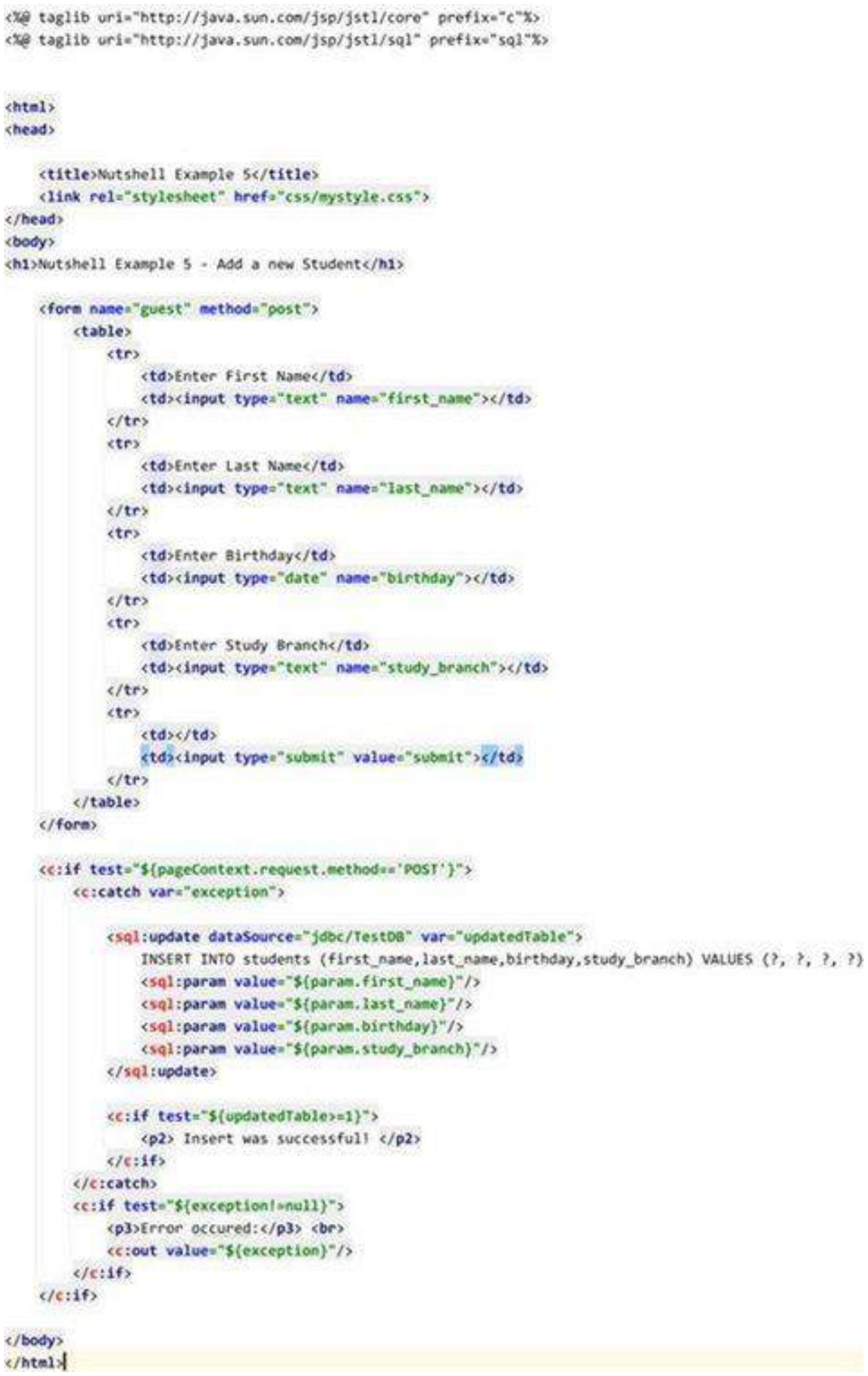
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Appetizer</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>
<body>

<h1>Dessert</h1>

<ul>

    <li><a href="Nutshell5-InsertData.jsp">"Nutshell Example 5-InsertData.jsp"</a>
        insert a student into the database
    </li>
    <li><a href="Nutshell6-SelectData.jsp">"Nutshell Example 6-SelectData.jsp"</a>
        select all students from the database
    </li>
    <li><a href="Nutshell7-UpdateData.jsp">"Nutshell Example 7-UpdateData.jsp"</a>
        update the study branch of a student
    </li>
    <li><a href="Nutshell8-DeleteData.jsp">"Nutshell Example 8-DeleteData.jsp"</a>
        delete a student from the database table
    </li>
    <li><a href="Nutshell9-GroupWorkDivider.jsp">"Nutshell Example 9-GroupWorkDivider.jsp"</a>
        divides all students in working groups
    </li>

</ul>

</body>
</html>
```

## Dessert/WEB-INF/web.xml



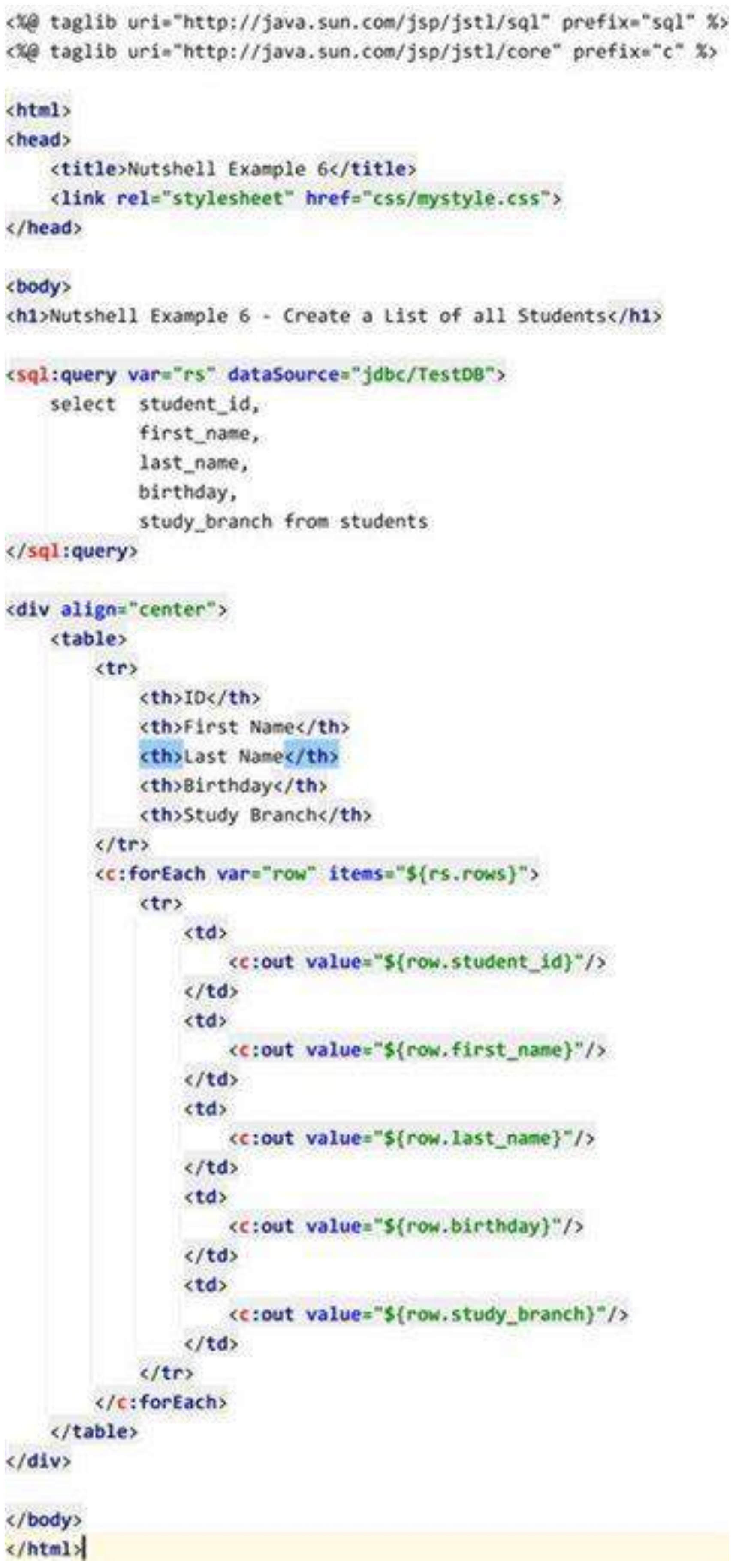
```xml
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">


    <description>MySQL Test App</description>
    <resource-ref>
        <description>DB Connection</description>
        <res-ref-name>jdbc/TestDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>

    <servlet>
        <servlet-name>cgi</servlet-name>
        <servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>
        <init-param>
            <param-name>cgiPathPrefix</param-name>
            <param-value>WEB-INF/cgi</param-value>
        </init-param>
        <init-param>
            <param-name>executable</param-name>
            <param-value>rexx</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>cgi</servlet-name>
        <url-pattern>/cgi-bin/*</url-pattern>
    </servlet-mapping>


</web-app>
```

## Nutshell5 - InsertData



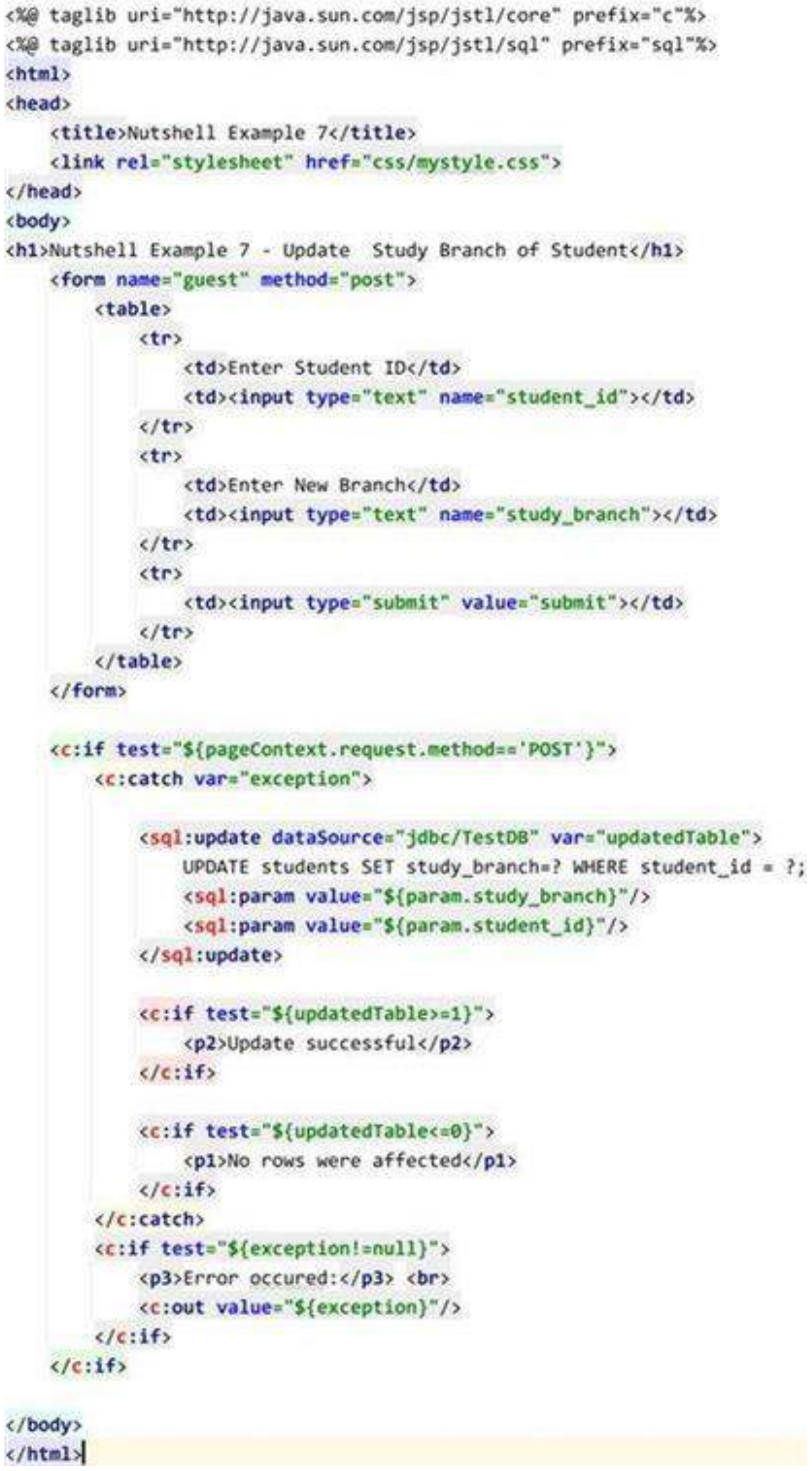
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>


<html>
<head>

    <title>Nutshell Example 5</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>
<body>
<h1>Nutshell Example 5 - Add a new Student</h1>

    <form name="guest" method="post">
        <table>
            <tr>
                <td>Enter First Name</td>
                <td><input type="text" name="first_name"></td>
            </tr>
            <tr>
                <td>Enter Last Name</td>
                <td><input type="text" name="last_name"></td>
            </tr>
            <tr>
                <td>Enter Birthday</td>
                <td><input type="date" name="birthday"></td>
            </tr>
            <tr>
                <td>Enter Study Branch</td>
                <td><input type="text" name="study_branch"></td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="submit"></td>
            </tr>
        </table>
    </form>

    <c:if test="${pageContext.request.method=='POST'}">
        <c:catch var="exception">

            <sql:update dataSource="jdbc/TestDB" var="updatedTable">
                INSERT INTO students (first_name,last_name,birthday,study_branch) VALUES (?, ?, ?, ?)
                <sql:param value="${param.first_name}"/>
                <sql:param value="${param.last_name}"/>
                <sql:param value="${param.birthday}"/>
                <sql:param value="${param.study_branch}"/>
            </sql:update>

            <c:if test="${updatedTable>=1}">
                <p2> Insert was successful! </p2>
            </c:if>
        </c:catch>
        <c:if test="${exception!=null}">
            <p3>Error occured:</p3> <br>
            <c:out value="${exception}"/>
        </c:if>
    </c:if>

</body>
</html>
```

## Nutshell6 – SelectData



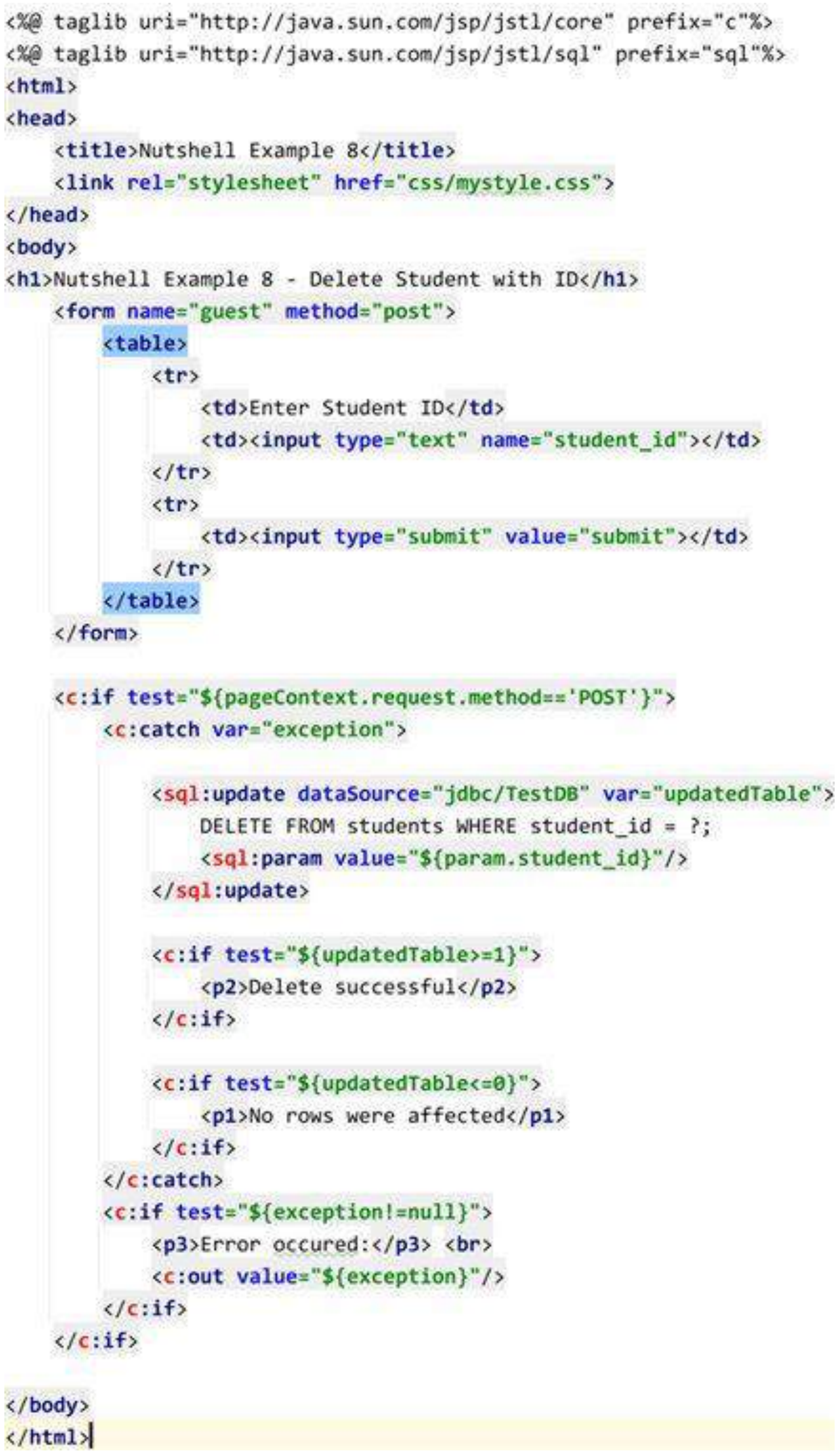
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>
<head>
    <title>Nutshell Example 6</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>

<body>
<h1>Nutshell Example 6 - Create a List of all Students</h1>

<sql:query var="rs" dataSource="jdbc/TestDB">
    select  student_id,
            first_name,
            last_name,
            birthday,
            study_branch from students
</sql:query>

<div align="center">
    <table>
        <tr>
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Birthday</th>
            <th>Study Branch</th>
        </tr>
        <c:forEach var="row" items="${rs.rows}">
            <tr>
                <td>
                    <c:out value="${row.student_id}"/>
                </td>
                <td>
                    <c:out value="${row.first_name}"/>
                </td>
                <td>
                    <c:out value="${row.last_name}"/>
                </td>
                <td>
                    <c:out value="${row.birthday}"/>
                </td>
                <td>
                    <c:out value="${row.study_branch}"/>
                </td>
            </tr>
        </c:forEach>
    </table>
</div>

</body>
</html>
```

## Nutshell7 – UpdateData.jsp



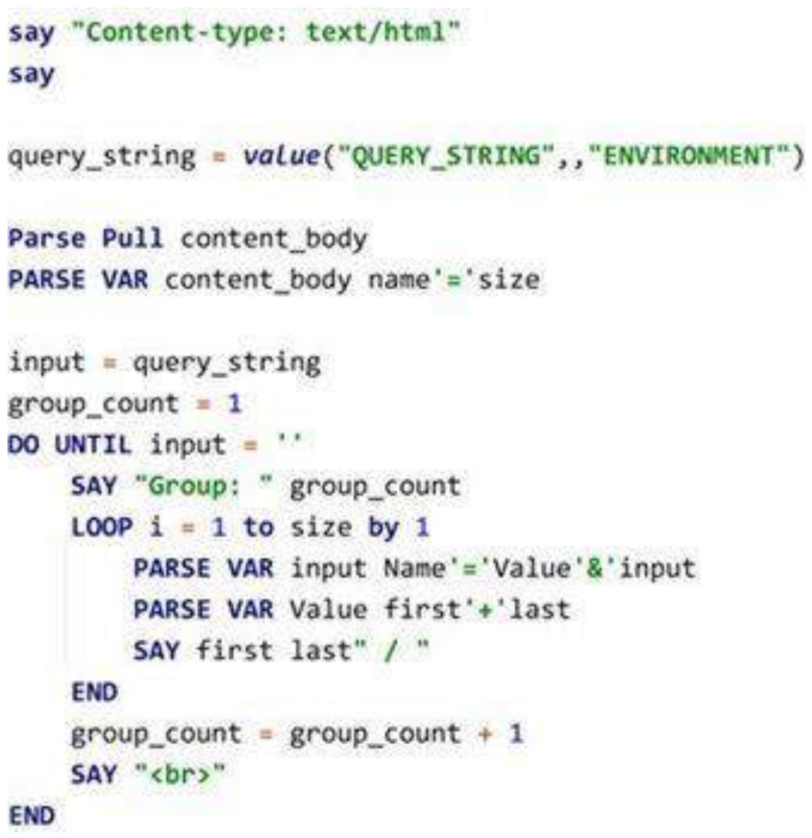
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
    <title>Nutshell Example 7</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>
<body>
<h1>Nutshell Example 7 - Update  Study Branch of Student</h1>
    <form name="guest" method="post">
        <table>
            <tr>
                <td>Enter Student ID</td>
                <td><input type="text" name="student_id"></td>
            </tr>
            <tr>
                <td>Enter New Branch</td>
                <td><input type="text" name="study_branch"></td>
            </tr>
            <tr>
                <td><input type="submit" value="submit"></td>
            </tr>
        </table>
    </form>

    <c:if test="${pageContext.request.method=='POST'}">
        <c:catch var="exception">

            <sql:update dataSource="jdbc/TestDB" var="updatedTable">
                UPDATE students SET study_branch=? WHERE student_id = ?;
                <sql:param value="${param.study_branch}"/>
                <sql:param value="${param.student_id}"/>
            </sql:update>

            <c:if test="${updatedTable>=1}">
                <p2>Update successful</p2>
            </c:if>

            <c:if test="${updatedTable<=0}">
                <p1>No rows were affected</p1>
            </c:if>
        </c:catch>
        <c:if test="${exception!=null}">
            <p3>Error occured:</p3> <br>
            <c:out value="${exception}"/>
        </c:if>
    </c:if>

</body>
</html>
```

## Nutshell8 – DeleteData.jsp



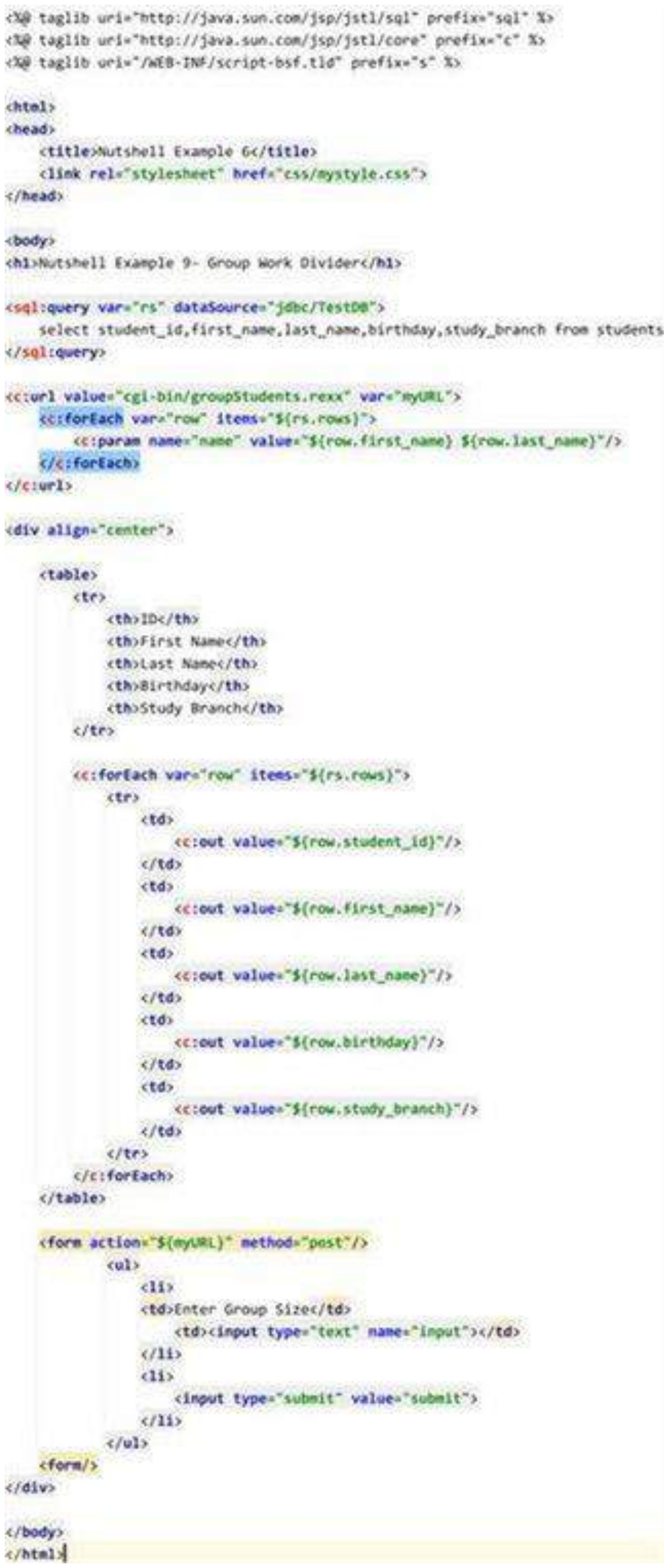
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
    <title>Nutshell Example 8</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>
<body>
<h1>Nutshell Example 8 - Delete Student with ID</h1>
    <form name="guest" method="post">
        <table>
            <tr>
                <td>Enter Student ID</td>
                <td><input type="text" name="student_id"></td>
            </tr>
            <tr>
                <td><input type="submit" value="submit"></td>
            </tr>
        </table>
    </form>

    <c:if test="${pageContext.request.method=='POST'}">
        <c:catch var="exception">

            <sql:update dataSource="jdbc/TestDB" var="updatedTable">
                DELETE FROM students WHERE student_id = ?;
                <sql:param value="${param.student_id}"/>
            </sql:update>

            <c:if test="${updatedTable>=1}">
                <p2>Delete successful</p2>
            </c:if>

            <c:if test="${updatedTable<=0}">
                <p1>No rows were affected</p1>
            </c:if>
        </c:catch>
        <c:if test="${exception!=null}">
            <p3>Error occured:</p3> <br>
            <c:out value="${exception}"/>
        </c:if>
    </c:if>

</body>
</html>
```

## Dessert/WEB-INF/cgi/groupStudents.rex



```
say "Content-type: text/html"
say

query_string = value("QUERY_STRING",,"ENVIRONMENT")

Parse Pull content_body
PARSE VAR content_body name'='size

input = query_string
group_count = 1
DO UNTIL input = ''
    SAY "Group: " group_count
    LOOP i = 1 to size by 1
        PARSE VAR input Name'='Value'&'input
        PARSE VAR Value first'+'last
        SAY first last" / "
    END
    group_count = group_count + 1
    SAY "<br>"
END
```

## Nutshell 9 - GroupWorkDivider



```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="/WEB-INF/script-bsf.tld" prefix="s" %>

<html>
<head>
    <title>Nutshell Example 6</title>
    <link rel="stylesheet" href="css/mystyle.css">
</head>

<body>
<h1>Nutshell Example 9- Group Work Divider</h1>

<sql:query var="rs" dataSource="jdbc/TestDB">
    select student_id,first_name,last_name,birthday,study_branch from students
</sql:query>

<c:url value="cgi-bin/groupStudents.rexx" var="myURL">
    <c:forEach var="row" items="${rs.rows}">
        <c:param name="name" value="${row.first_name} ${row.last_name}"/>
    </c:forEach>
</c:url>

<div align="center">

    <table>
        <tr>
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Birthday</th>
            <th>Study Branch</th>
        </tr>

        <c:forEach var="row" items="${rs.rows}">
            <tr>
                <td>
                    <c:out value="${row.student_id}"/>
                </td>
                <td>
                    <c:out value="${row.first_name}"/>
                </td>
                <td>
                    <c:out value="${row.last_name}"/>
                </td>
                <td>
                    <c:out value="${row.birthday}"/>
                </td>
                <td>
                    <c:out value="${row.study_branch}"/>
                </td>
            </tr>
        </c:forEach>
    </table>

    <form action="${myURL}" method="post"/>
            <ul>
                <li>
                <td>Enter Group Size</td>
                    <td><input type="text" name="input"></td>
                </li>
                <li>
                    <input type="submit" value="submit">
                </li>
            </ul>
    <form/>
</div>

</body>
</html>
```

## Bibliography

[1] Tomcat Software Foundation - Apache Tomcat Homepage; Retrieved 11.10.2020, from https://tomcat.apache.org/index.html

[2] Flatscher, R. G. (2016). WU-Wien/BIS-Kurs4; Retrieved 11.10.2020, from

http://wi.wu.ac.at:8002/rgf/wu/lehre/autojava/material/foils/AutoJava-BSF4ooRexx-01.pdf

[3] BSF4ooRexx - Installation Guide, from BSF4ooRexx Installation %PATH%\bsf4oorexx\information\installation

[4] Tomcat Software Foundation - „Apache Tomcat Wiki"– Configure Manager Application; Retrieved 20.10.2020 from https://tomcat.apache.org/tomcat-8.0-doc/manager-howto.html#Configuring_Manager_Application_Access

[5] Tutorials Point – JSP Tutorial; Retrieved 06.11.2020, from https://www.tutorialspoint.com/jsp/index.htm

[6] Tutorials Point – PHP GET&POST Methods; Retrieved 22.11.2020, from https://www.tutorialspoint.com/php/php_get_post.htm

[7] Apache Software Foundation – CGI How To; Retrieved 28.11.2020, from https://tomcat.apache.org/tomcat-8.0-doc/cgi-howto.html

[8] W3SCHOOLS – How To Add CSS – Retrieved 06.12.2020, from https://www.w3schools.com/css/css_howto.asp

[9] JavaTPoint – JSTL (Standard Tag Library); Retrieved 02.12.2020, https:/www.javatpoint.com/jstl

[10] MySQL Workbench Manual – Developer Guide Installation; Retrieved 02.12.2020, from https://dev.mysql.com/doc/workbench/en/wb-installing-windows.html

[11] Tutorials Point – JSTL Core <c:param> Tag; Retrieved 06.12.2020, from https://www.tutorialspoint.com/jsp/jstl_core_param_tag.htm