Web Application Programmer's Guide

1.1 Audience and Content

The audience for this guide is the Enterprise Bean provider, i.e. the person in charge of developing the software components on the server side and, more specifically, the web components.

1.2 Developing Web Components

1.2.1 Introduction

A Web Component is a generic term which denotes both JSP pages and Servlets. Web Components are packaged in a .war file and can be deployed to a server via the web container service. Web Components can be integrated in a J2EE application by packing the .war file in an .ear file.

The directory structure of this application is the following:

F16g324/WebContent/WEB-INF/web.xml	Contains the web.xml file describing the web application
F16g324/WebContent/jsp	Contains the JSP pages and images
F16g324/src/edu/uic/controller	Servlet sources
F16g324/src/edu/model/bean	Bean sources

The bean directory can also be included as an external bean directory if coming from another application.

1.2.2 The JSP Pages

Java Server Pages(JSP) is a technology that allows regular, static HTML, to be mixed with dynamically-generated HTML written in a Java programming language for encapsulating the logic that generates the content for the page.

1.2.2.1 Example:

The following example shows a sample JSP page that lists the content of the web application.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<%@ taglib prefix="t" uri="http://myfaces.apache.org/tomahawk"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>OTTS</title>
<style>
.ui-input-invalid {
        font-color: red
}
</style>
<f:view>
        <center>
                <header>
                        <h2>f16g324 Online Test Taking System</h2>
                </header>
        </center>
        <center>
                <h3>Login</h3>
        </center>
        <hr>
        <center>
                <h:form>
                        <div style="text-align: right">
                                   
                                <h:commandLink action="/doc/UserGuide.pdf" value="User Guide"></h:commandLink>
                                   
                                <h:commandLink action="/jsp/AboutUs.jsp" value="About Us"></h:commandLink>
                        </div>
                        <br>
                        <br>
                        <br>
                        <br>
                        <h:messages globalOnly="false" layout="table" style="color:red" />
                        <table>
                                <tr>
                                        <td><h:outputText value="Username" /></td>
                                        <td><h:inputText id="username" value="#{LoginUserBean.user}"
                                                        required="true" requiredMessage="Please enter the username">
                                                </h:inputText></td>
                                </tr>
                                <tr>
                                        <td><h:outputText value="Password" /></td>
                                        <td><h:inputSecret id="password" value="#{LoginUserBean.pwd}"
                                                        required="true" requiredMessage="Please enter the password"></h:inputSecret></td>
                                </tr>
                                <tr>
                                        <td><h:outputText value="Role" /></td>
                                        <td><h:selectOneMenu value="#{LoginUserBean.role}">
                                                        <f:selectItem itemValue="Student" />
                                                        <f:selectItem itemValue="Teacher" />
                                                        <f:selectItem itemValue="Admin" />
                                                </h:selectOneMenu></td>
                                </tr>
                        </table>
                        <br />
                        <h:commandButton action="#{LoginUserBean.validateUsernamePassword}"
                                value="Login"></h:commandButton>
                </h:form>
        </center>
</f:view>
</html>
```

1.2.3. The Servlets

Servlets are modules of Java code that run in an application server for answering client requests. Servlets are not tied to a specific client -server protocol. However, they are most commonly used with HTTP, and the word “Servlet” is often used as referring to an “HTTP Servlet”.
Servlets make use of the Java standard extension classes in the packages javax.servlet and javax.servlet.http.

Typical uses for HTTP Servlets include:

- processing and/or storing data submitted by an HTML form,
- providing dynamic content generated by processing a database query,
- managing information of the HTTP request.

1.2.3.1 Example:

The following example is a sample of a Servlet that lists the content of a cart. This example is the servlet version of the previous JSP page example.

```
package edu.uic.controller;


import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import edu.uic.dao.DBDao;
import edu.uic.dao.DBService;

public class Login extends HttpServlet {

//      private static final long serialVersionUID = 1L;

        private final static String mysqlJdbcDriver = "com.mysql.jdbc.Driver";

        private static String dbUserNameForTestSchema = "s16g40";
        private static String dbPasswordForTestSchema = "s16g40FpqU5";
        private static String testSchema = "s16g40";

        public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String userName = request.getParameter("username");
                String password = request.getParameter("password");

                try {
                        DBDao dao1 = new DBDao(dbUserNameForTestSchema, dbPasswordForTestSchema,
                                        "jdbc:mysql://131.193.209.57:3306/", testSchema, mysqlJdbcDriver);
                        dao1.createConnection();
                        DBService serv = new DBService(dao1);
                        String wherClause = "username = '" + userName + "'";
                        ResultSet resultSet = serv.fetchDataFromTable("f16g324_user", wherClause);
                        boolean returnType = false;
                        int role = 1;
                        while (resultSet.next()) {
                                if (resultSet.getString(3).equals(password)) {
                                        role = Integer.parseInt(resultSet.getString(6));
                                        returnType = true;
                                }
                        }
                        dao1.closeConnection();
                        if (!returnType)
                                request.getRequestDispatcher("//jsp/LoginFailure.jsp").forward(request, response);
                        else if(role==1)
                                request.getRequestDispatcher("//jsp/StudentHome.jsp").forward(request, response);
                        else if(role==2)
                                request.getRequestDispatcher("//jsp/TeacherHome.jsp").forward(request, response);
                } catch (Exception e2) {
                        System.out.println(e2);
                }

                out.close();
        }
}
```

1.2.4 JSF

JSF is a component based technology that allows you to develop web applications. It is expandable and extensible with complex, built-in, JavaScript-driven components. JSF is a specification that is implemented by JSF providers such as Oracle and the Apache Software Foundation. Third-party component libraries can add significant functionality to the core JSF components.

In JSF, the annotation ManagedBean indicates that the Java class or bean can be accessed by any JSF page.

1.2.4.1 Example

The following example is a sample JSF page that lists the content of the web app.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<%@ taglib prefix="t" uri="http://myfaces.apache.org/tomahawk"%>

<f:view>
        <center>
                <header>
                        <h2>f16g324 Online Test Taking System</h2>
                </header>
        </center>
        <center>
                <h3>Admin Home</h3>
        </center>
        <hr>
        <h:form>

        Welcome <%=session.getAttribute("firstname")%>


                <div style="text-align: right;">
                          
                        <h:commandLink action="#{LoginUserBean.logout}" value="Logout"
                                style="text-align: right"></h:commandLink>
                </div>

                <center>
                        List of available Courses <br> <br> <br>

                        <t:dataTable value="#{CourseBean.courseBeanList}" var="rowNumber"
                                rendered="true" border="1" cellspacing="0" cellpadding="1"
                                columnClasses="columnClass1 border" headerClass="headerClass"
                                footerClass="footerClass" rowClasses="rowClass2"
                                styleClass="dataTableEx" width="800">
                                <h:column>
                                        <f:facet name="header">
                                                <h:outputText>CRN#</h:outputText>
                                        </f:facet>
                                        <h:outputText value="#{rowNumber.crn}" />
                                </h:column>
                                <h:column>
                                        <f:facet name="header">
                                                <h:outputText>Course Code</h:outputText>
                                        </f:facet>
                                        <h:outputText value="#{rowNumber.courseCode}" />
                                </h:column>
                                <h:column>
                                        <f:facet name="header">
                                                <h:outputText>Course Name</h:outputText>
                                        </f:facet>
                                        <h:outputText value="#{rowNumber.coursename}" />
                                </h:column>
                                <h:column>
                                        <f:facet name="header">
                                                <h:outputText>
                                Course Description</h:outputText>
                                        </f:facet>
                                        <h:outputText value="#{rowNumber.coursedescription}" />

                                </h:column>

                        </t:dataTable>
                </center>
```

```
                </center>
        </h:form>
        <center>
                <br>
                <br>
                <br>
                <h:form enctype="multipart/form-data">
                        Upload Test Roaster :

                                <t:inputFileUpload id="file" value="#{CourseBean.uploadedFile}"
                                        required="true" />

                                <h:commandButton value="Submit" action="#{CourseBean.submit}" />

                </h:form>
        </center>
</f:view>
```

1.2.4.2 Examples

The following example is sample JSF page that lists the content of the web app.

```
package edu.uic.model.bean;

import edu.uic.dao.DBDao;
import edu.uic.dao.DBService;
import org.apache.commons.io.IOUtils;
import org.apache.myfaces.custom.fileupload.UploadedFile;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import java.io.*;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

@ManagedBean
@SessionScoped
public class TestBean extends QuestionBean implements Serializable {
        private UploadedFile uploadedFile;
    private String fileName;
        private static final long serialVersionUID = 1094801825228386363L;
        private int testNumber;
        private int crn;
        private int courseCode;
        private String startDate;
        private String endDate;
        private Boolean isButtonRendered ;

        private String t;

        public String getT() {
                return t;
        }

        public void setT(String t) {
                this.t = t;
        }

        public Boolean getIsButtonRendered() {
                return isButtonRendered;
        }

        public void setIsButtonRendered(Boolean isButtonRendered) {
                this.isButtonRendered = isButtonRendered;
        }

        private List<TestBean> testBeanList;

        private static final String dbUserNameForTestSchema = "f16g324";
        private static final String dbPasswordForTestSchema = "g324Xp1bjr";
        private static final String mysqlJdbcDriver = "com.mysql.jdbc.Driver";
        private static final String testSchema = "f16g324";

        public int getTestNumber() {
                return testNumber;
        }

        public void setTestNumber(int testNumber) {
                this.testNumber = testNumber;
        }
```

```
public int getCrn() {
	return crn;
}

public void setCrn(int crn) {
	this.crn = crn;
}

public int getCourseCode() {
	return courseCode;
}

public void setCourseCode(int courseCode) {
	this.courseCode = courseCode;
}

public String getStartDate() {
	return startDate;
}

public void setStartDate(String startDate) {
	this.startDate = startDate;
}

public String getEndDate() {
	return endDate;
}

public void setEndDate(String endDate) {
	this.endDate = endDate;
}

public List<TestBean> getTestBeanList() {
	return testBeanList;
}

public void setTestBeanList(List<TestBean> testBeanList) {
	this.testBeanList = testBeanList;
}

public TestBean(int k) {
}

public TestBean() {
	DBDao dao1 = new DBDao(dbUserNameForTestSchema, dbPasswordForTestSchema, "jdbc:mysql://131.193.209.57:3306/",
			testSchema, mysqlJdbcDriver);
	try {

		dao1.createConnection();
		if (dao1.getConnection() != null) {
			DBService serv = new DBService(dao1);
			ResultSet resultSet = serv.fetchDataFromTable("test", -1, -1);
			this.testBeanList = new ArrayList<TestBean>();
			TestBean b = new TestBean(1);
			while (resultSet.next()) {
				if (resultSet.getString(1) != null) {
					b.setTestNumber(Integer.parseInt(resultSet.getString(1)));
					b.setCrn(Integer.parseInt(resultSet.getString(2)));
					b.setCourseCode(Integer.parseInt(resultSet.getString(3)));
					b.setStartDate(resultSet.getString(4));
					b.setEndDate(resultSet.getString(5));
					boolean isAvailable = checkForTest(serv, Integer.parseInt(resultSet.getString(1)),Integer.parseInt(resultSet.getString(6)));
					b.setIsButtonRendered(isAvailable);
					this.testBeanList.add(b);
```

```
private boolean checkForTest(DBService serv, int i, int j) throws SQLException {
        while (serv.fetchDataFromTable("question", "testid=" + i).next()) {
                                return true;
        }
        return false;
}

public String takeTest() {
        return "success";
}
public String submit() {

        DBDao dao1 = new DBDao(dbUserNameForTestSchema, dbPasswordForTestSchema, "jdbc:mysql://131.193.209.57:3306/",
                        testSchema, mysqlJdbcDriver);



File file = null;
OutputStream output = null;
String returnMSG = "success";
try {
    BufferedReader reader = new BufferedReader(new InputStreamReader(uploadedFile.getInputStream()));
    StringBuilder out = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        out.append(line);
    }
    System.out.println(out.toString());   //Prints the string content read from input stream
    reader.close();
```

1.3 Defining the Web Deployment Descriptor

1.3.1 Principles

The web component programmer is responsible for providing the deployment descriptor with the developed web components. The Web component provider's responsibilities and the application assembler's responsibilities are to provide the XML deployment descriptor that conforms to the deployment descriptor's XML Schema as defined in the Java Servlet Specification.

To customize the Web Components, information not defined in the standard XML deployment descriptor may be needed. For example, the information may include the mapping of the name of referenced resources to its JDNI name. This information can be specified during the deployment phase, within another XML deployment descriptor that is specific.

The parser gets the specified schema via the classpath (schemas are packaged in .jar file).

The standard deployment descriptor(web.xml) should contain structural information that includes the following:

- The Servlet's description (including Servlet's name, Servlet's class or jsp-file .Servlet's initialization parameters),
- Environment entries,
- EJB references,
- EJB local references,
- Resource references,
- Resource env references.

<host> element: If the configuration file of the web container contains virtual hosts, the host on which the WAR file is deployed can be set.

<context-root> element: The name of the context on which the application will be deployed should be specified. If it is not specified, the context-root used can be one of the following:

- If the war is packaged into an EAR file, the context-root used is the context specified in the application .xml file.
- If the war is standalone, the context-root is the name of the war file

If the context-root is/or empty, the web application is deployed as ROOT context.

<java2-delegation-model> element: Set the compliance to the Java 2 delegation model.

- If true: the web application context uses a class loader, using the Java 2 delegation model.
- If false: the class loader searches inside the web application first, before asking parent class loaders.

1.3.2. Examples of Web Deployment Descriptors

Example of a standard Web Deployment Descriptor (web.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         id="WebApp_ID" version="3.0">
  <display-name>Online Test Taking System</display-name>
  <welcome-file-list>
    <welcome-file>faces/jsp/login.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>Register</servlet-name>
    <servlet-class>edu.uic.controller.Register</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>edu.uic.controller.Login</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>AddTest</servlet-name>
    <servlet-class>edu.uic.controller.AddTest</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>ViewStudents</servlet-name>
    <servlet-class>edu.uic.controller.ViewStudents</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>TakeTest</servlet-name>
    <servlet-class>edu.uic.controller.TakeTest</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>ViewScores</servlet-name>
    <servlet-class>edu.uic.controller.ViewScores</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Register</servlet-name>
    <url-pattern>/servlet/Register</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/servlet/Login</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>TakeTest</servlet-name>
    <url-pattern>/servlet/TakeTest</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ViewScores</servlet-name>
    <url-pattern>/servlet/ViewScores</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>AddTest</servlet-name>
    <url-pattern>/servlet/AddTest</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ViewStudents</servlet-name>
    <url-pattern>/servlet/ViewStudents</url-pattern>
```

```
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
    <param-value>resources.application</param-value>
  </context-param>
  <context-param>
    <description>State saving method: 'client' or 'server' (=default). See JSF Specification 2.5.2</description>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
  </context-param>
  <context-param>
    <description>
        This parameter tells MyFaces if javascript code should be allowed in
        the rendered HTML output.
        If javascript is allowed, command_link anchors will have javascript code
        that submits the corresponding form.
        If javascript is not allowed, the state saving info and nested parameters
        will be added as url parameters.
        Default is 'true'</description>
    <param-name>org.apache.myfaces.ALLOW_JAVASCRIPT</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <description>
        If true, rendered HTML code will be formatted, so that it is 'human-readable'
        i.e. additional line separators and whitespace will be written, that do not
        influence the HTML code.
        Default is 'true'</description>
    <param-name>org.apache.myfaces.PRETTY_HTML</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <param-name>org.apache.myfaces.DETECT_JAVASCRIPT</param-name>
    <param-value>false</param-value>
  </context-param>
  <context-param>
    <description>
        If true, a javascript function will be rendered that is able to restore the
        former vertical scroll on every request. Convenient feature if you have pages
        with long lists and you do not want the browser page to always jump to the top
        if you trigger a link or button action that stays on the same page.
        Default is 'false'
</description>
    <param-name>org.apache.myfaces.AUTO_SCROLL</param-name>
    <param-value>true</param-value>
  </context-param>
  <listener>
    <listener-class>org.apache.myfaces.webapp.StartupServletContextListener</listener-class>
  </listener>
</web-app>
```

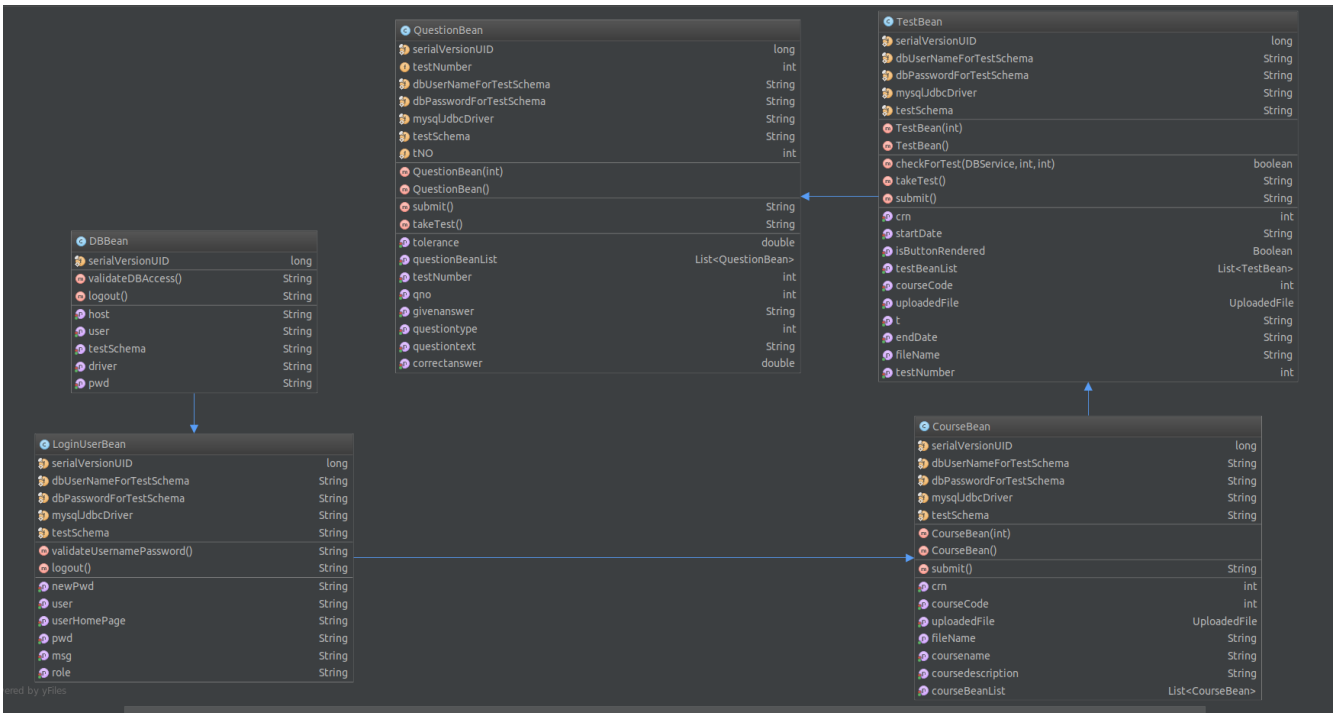
1.4 UML Diagrams

The UML diagrams acronym for Unified Modelling Language are a standard way to visualize and design of a system.

1.4.1 Examples

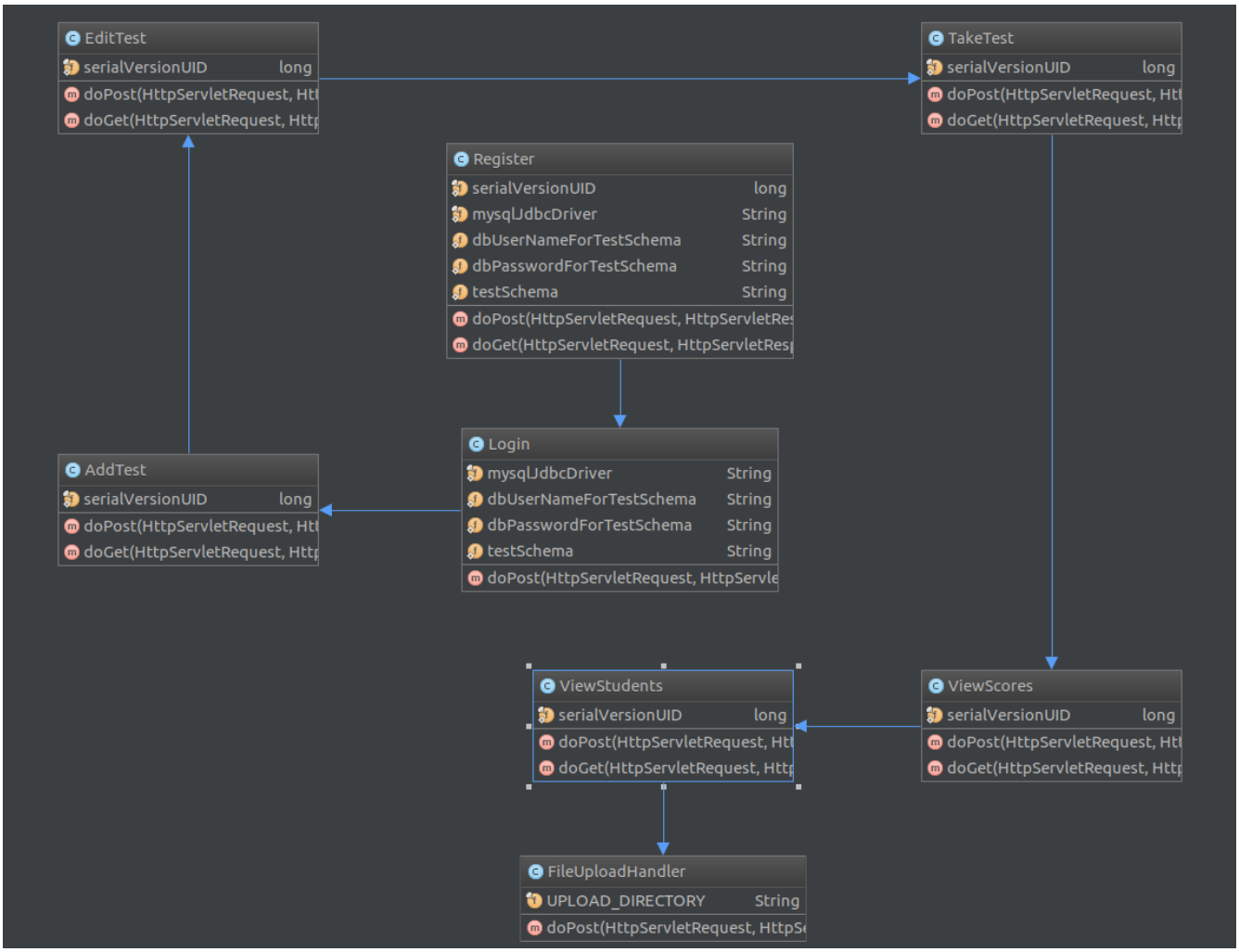
The following example is a sample UML diagram of the PackageBean that lists the contents of the all the Java classes within the bean package.

Every UML is connected to its peer as a dependency.



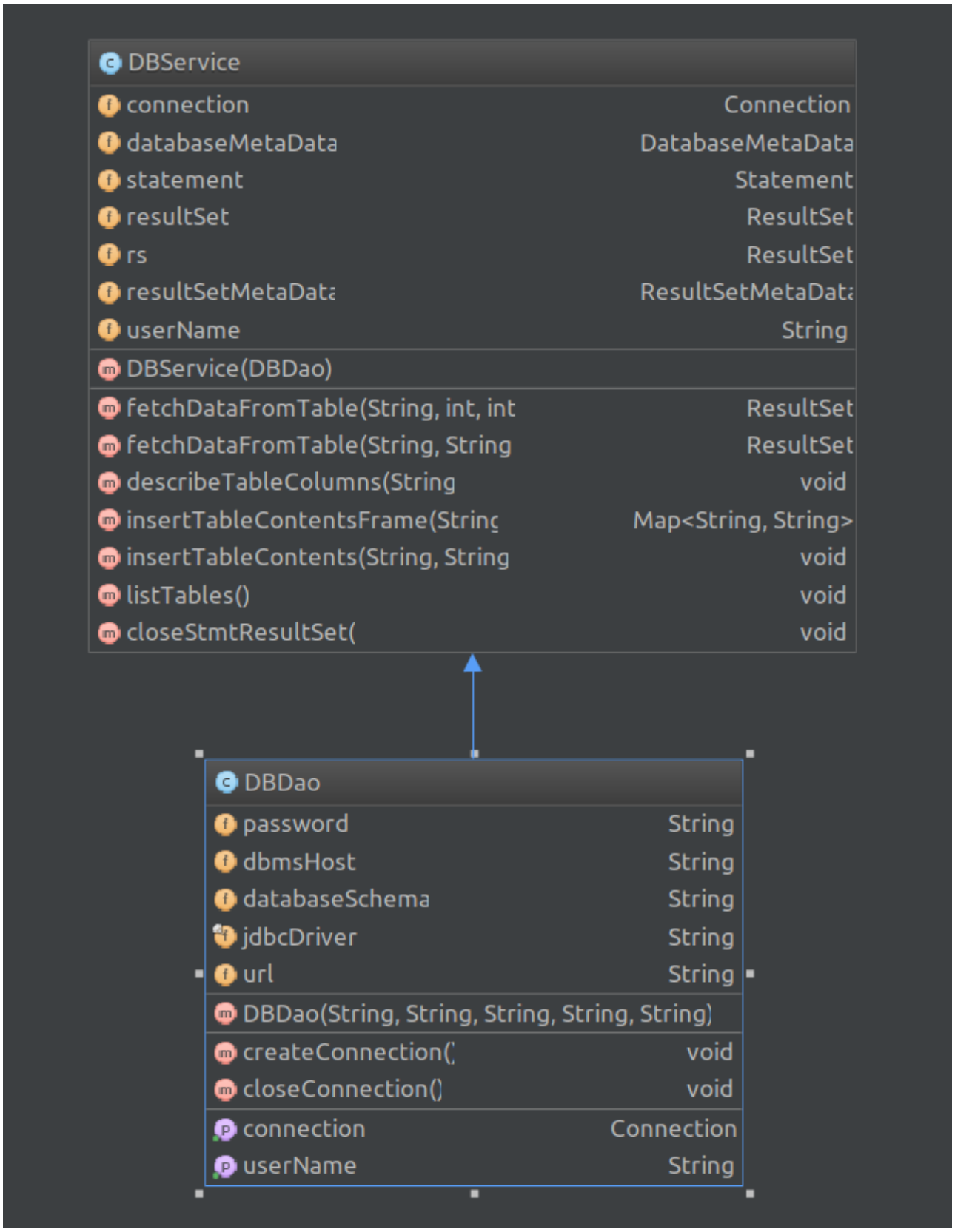
1.4.2 Example

The following example is UML diagram of the controller package that lists all the Servlets being employed in the project.


EditTest
serialVersionUID long
doPost(HttpServletRequest, Htt
doGet(HttpServletRequest, Http
TakeTest
serialVersionUID long
doPost(HttpServletRequest, Htt
doGet(HttpServletRequest, Http
Register
serialVersionUID long
mysqlJdbcDriver String
dbUserNameForTestSchema String
dbPasswordForTestSchema String
testSchema String
doPost(HttpServletRequest, HttpServletRes
doGet(HttpServletRequest, HttpServletResp
AddTest
serialVersionUID long
doPost(HttpServletRequest, Htt
doGet(HttpServletRequest, Http
Login
mysqlJdbcDriver String
dbUserNameForTestSchema String
dbPasswordForTestSchema String
testSchema String
doPost(HttpServletRequest, HttpServle
ViewStudents
serialVersionUID long
doPost(HttpServletRequest, Htt
doGet(HttpServletRequest, Http
ViewScores
serialVersionUID long
doPost(HttpServletRequest, Htt
doGet(HttpServletRequest, Http
FileUploadHandler
UPLOAD_DIRECTORY String
doPost(HttpServletRequest, HttpS

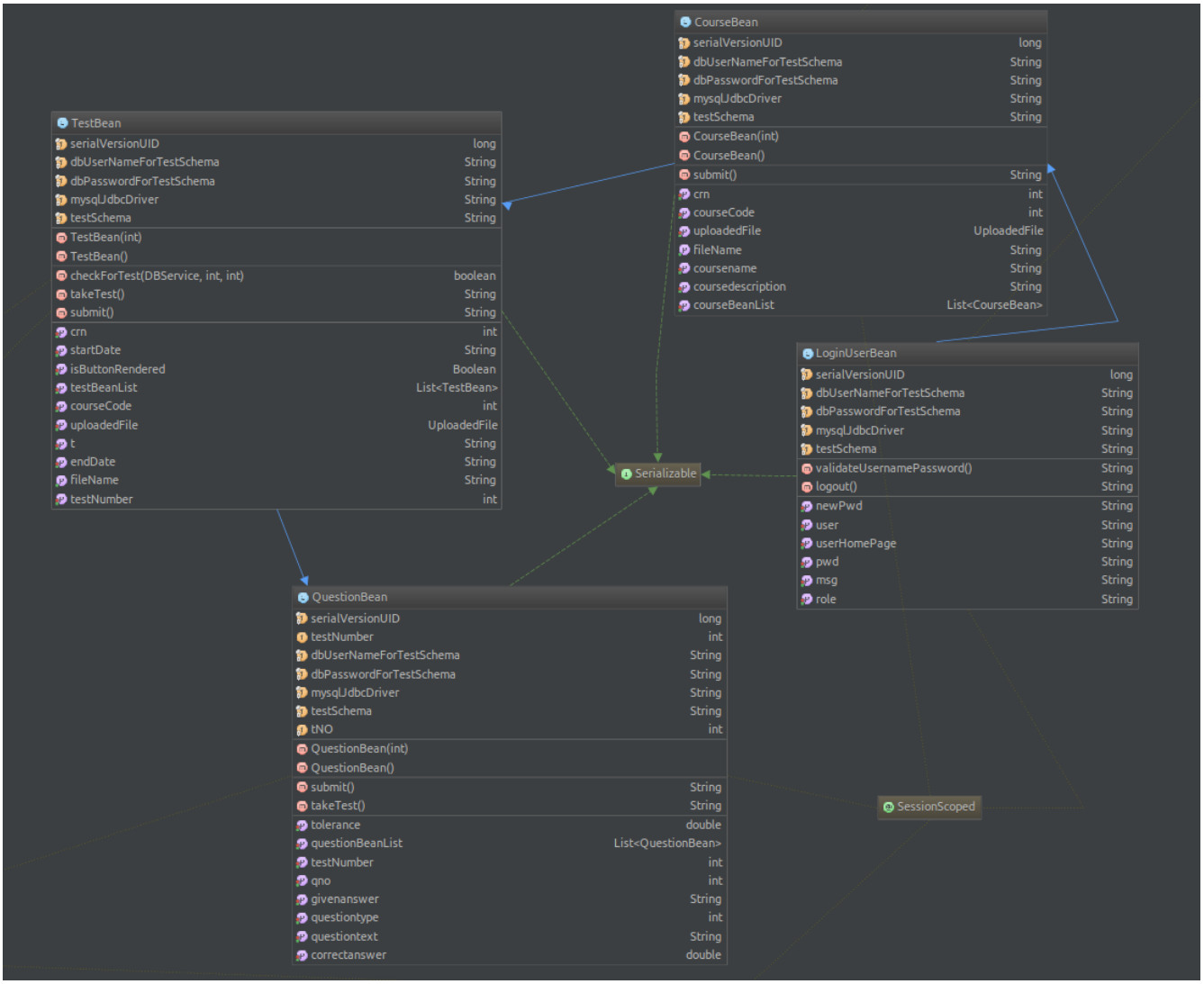
1.4.3 Examples

The following example is a UML diagram of the package Dao that lists the content which helps to connect to the db.



1.4.4 Example

The following example is a UML diagram of the LoginBean that lists the content of the login bean class.



1.5 WAR Packaging

Web Components are packaged for deployment in a standard Java Programming language Archive file called a war file (Web Archive), which is a jar similar to the package used for Java class libraries. A war has a specific hierarchical directory structure. The top-level directory of a war is the document root of the application.

The document root is where JSP pages, client-side classes and archives, and static web resources are stored. The document root contains a subdirectory called WEB-INF, which contains the following files and directories:

- web.xml: The standard xml deployment descriptor in the format defined in the Java Servlet 2.4 Specification.
- Classes: a directory that contains the servlet classes and utility classes.
- Lib: a directory that contains jar archives of libraries. If the Web application uses Enterprise Beans, it can also contain ejb-jars. This is necessary to give to the Web components the visibility of the EJB classes. However, if the war is intended to be packed in a ear, the ejb-jars must not be placed here. In this case, they are directly included in the ear. Due to the use of the class loader hierarchy, Web components have the visibility of the EJB classes.

1.5.1 Example

Before building a war file, the java source files must be complied to obtain the class files and the two XML deployment descriptor must be written.

Then, the war file is built using the jar command:

```
cd <your_webapp_directory>
jar cvf <web-application>.war *
```

During the development process, an ‘unpacked version’ of the war file can be used.

1.6 Web Service configuration

This service provides containers for the web components used by the Java EE applications.