Introduction to JSP

JavaServer Pages (JSP) technology enables you to mix regular, static HTML with dynamically generated content. You simply write the regular HTML in the normal manner, using familiar Web-page-building tools. You then enclose the code for the dynamic parts in special tags, most of which start with <% Java Code %>.

Servlets can be considered as Java code with HTML inside and JSP can be considered as HTML with Java code inside. Despite the large apparent differences between JSP pages and servlets, behind the scenes they are the same thing. JSP pages are translated into servlets, the servlets are compiled, and at request time it is the compiled servlets that execute. So, writing JSP pages is really just another way of writing servlets.

Need of JSP

Servlets have the following deficiencies when it comes to generating the output:

1- It is hard to write and maintain the HTML. Using print statements to generate HTML? Hardly convenient: you have to use parentheses and semicolons, have to insert backslashes in front of embedded double quotes, and have to use string concatenation to put the content together.

2- You cannot use standard HTML tools. All those great Web-site development tools you have are of little use when you are writing Java code.

3- The HTML is inaccessible to non-Java developers. If the HTML is embedded within Java code, a Web development expert who does not know the Java programming language will have trouble reviewing and changing the HTML.

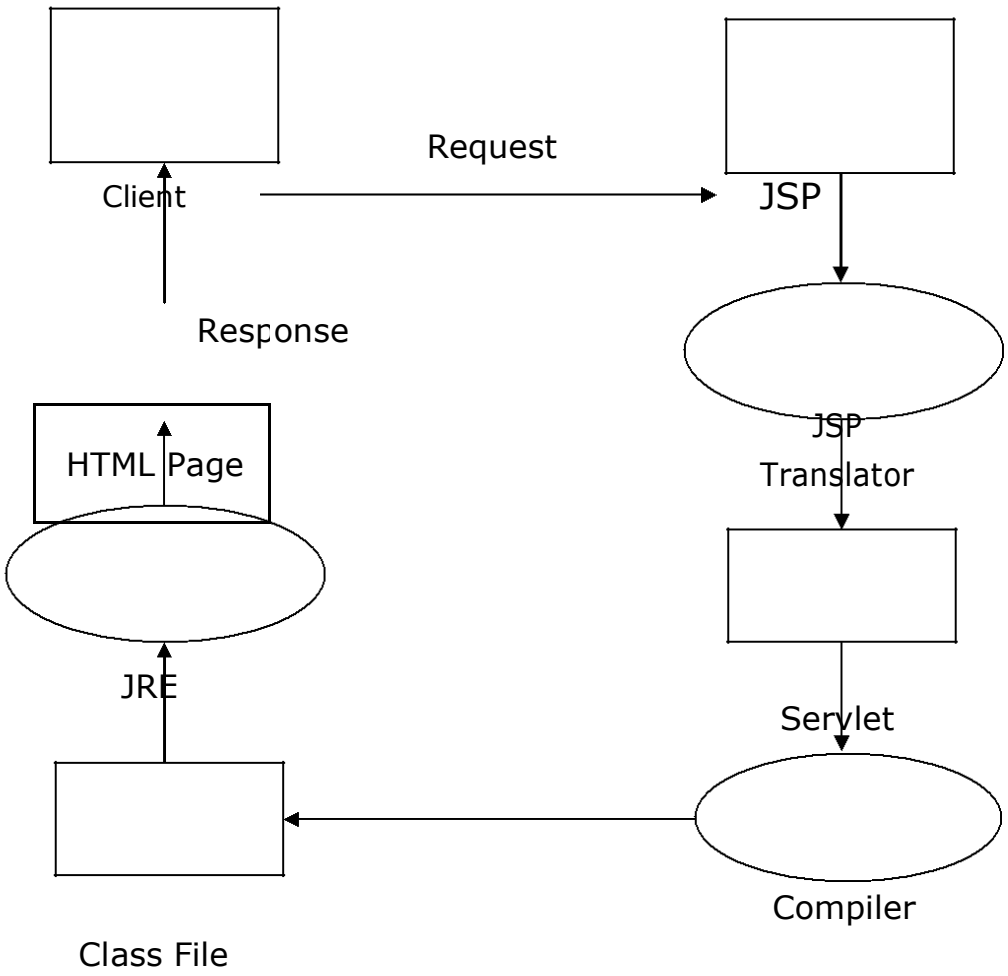
Example

```
<html>
      <body>
            <% out.print("Java Server Pages");
      %> </body>
</html>
```

The Lifecycle of a JSP Page

The JSP pages follow these phases:

1. Translation of JSP Page to Servlet.
2. Follow Servlet life cycle



A JSP page is converted into a servlet. The servlet is compiled, loaded into the server's memory, initialized, and executed. But which step happens when? Remember two points:

1- The JSP page is translated into a servlet and compiled only the first time it is accessed after having been modified.

2- Loading into memory, initialization, and execution follow the normal rules for servlets.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1- Extension to Servlet: - JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

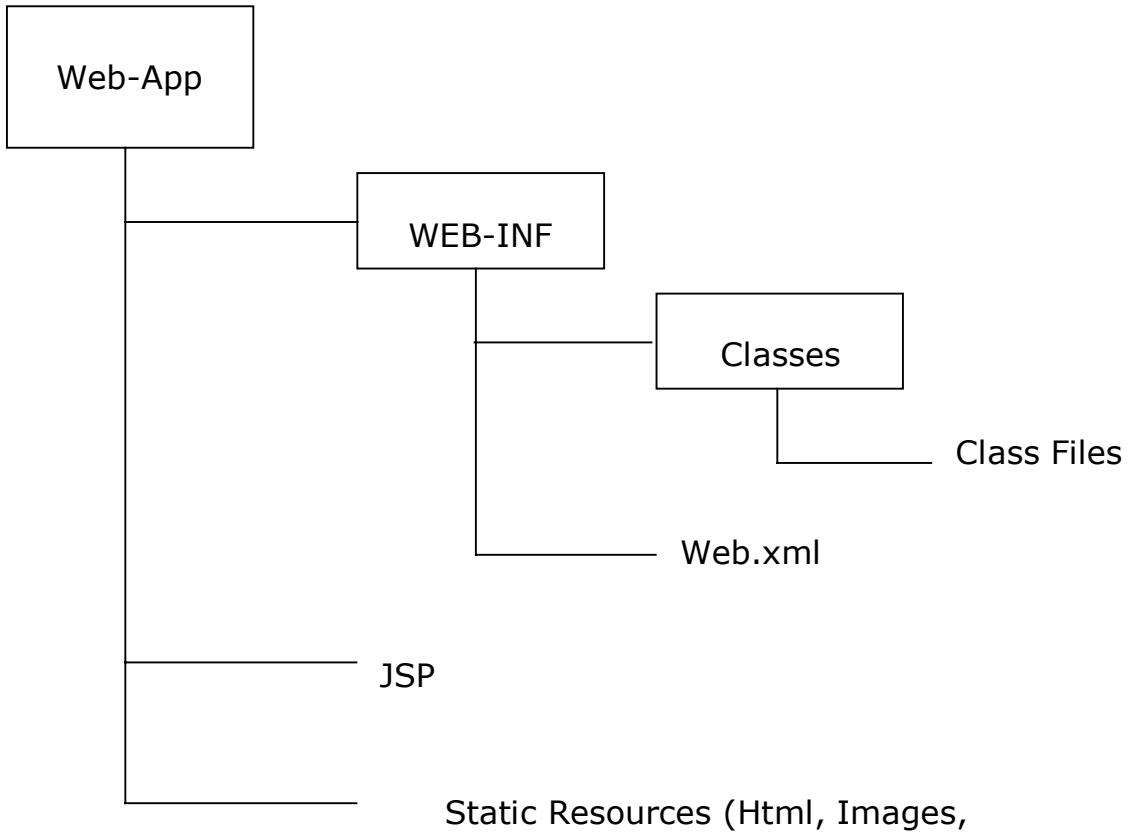
2- Easy to maintain: - JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3- Fast Development: - No need to recompile and redeploy: If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4- Less code than Servlet: - In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



JSP Scripting elements

The scripting elements provide the ability to insert java code inside the HTML Tags. There are three types of scripting elements:

1. Scriptlet tag.
2. Expression tag.
3. Declaration tag.

Scripting Element	Example
Comment	<%-- comment --%>
Directive	<%@ directive %>
Declaration	<%! declarations %>
Scriptlet	<% scriplets %>
Expression	<%= expression %>

Types of JSP Scripting Elements

JSP scripting elements let you insert Java code into the servlet that will be generated from the JSP page. There are three forms:

1- Expressions of the form, which are evaluated and inserted into the servlet's output.

2- Scriptlets of the form, which are inserted into the servlet's _jspService method (called by service).

3- Declarations of the form, which are inserted into the body of the servlet class, outside any existing methods.

JSP Expressions

A JSP expression is used to insert values directly into the output. It has the following form:

<% = Java Expression %>.

The expression is evaluated, converted to a string, and inserted in the page. This evaluation is performed at runtime (when the page is requested) and thus has full access to information about the request.

For example, the following shows the date/time that the page was requested. Current time: <% = java.util.Date() %>

Predefined Variables

To simplify these expressions, you can use a number of predefined variables (or "implicit objects"). The system simply tells you what names it will use for the

local variables in _jspService (the method that replaces doGet in servlets that result from JSP pages). These implicit objects are these:

1- request, the HttpServletRequest.

2- response, the HttpServletResponse.

3- session, the HttpSession associated with the request (unless disabled with the session attribute of the page directive—see Section 12.4).

4- out, the Writer (a buffered version of type JspWriter) used to send output to the client.

5- application, the ServletContext. This is a data structure shared by all servlets and JSP pages in the Web application and is good for storing shared data.

Host Name : <% = request.getRemoteHost() %>

JSP/Servlet Correspondence

JSP expressions basically become print (or write) statements in the servlet that results from the JSP page. Whereas regular HTML becomes print statements with double quotes around the text, JSP expressions become print statements with no double quotes. Instead of being placed in the doGet method, these print statements are placed in a new method called _jspService() that is called by service for both GET and POST requests.

FileName: LuckyNumber.jsp

```
<html>
  <body>
      Your Luck Number is: <% = Math.random() %>
  <body>
<html>
```

FileName: LuckyNumber.java

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

        response.setContentType("text/html");
```

```
            HttpSession session = request.getSession();
            JspWriter out = response.getWriter();

            out.println ("<html><body>”);
            out.println ("Your Luck Number is :");
            out.println (Math.random());
            out.println ("<body><html>”);
}
```

Servlet and Corresponding JSP

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ThreeParams extends HttpServlet
{
public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {

  response.setContentType("text/html");
  PrintWriter out = response.getWriter(); String
  title = "Reading Request Parameters";

  out.println(“<html><head><title>” + title + “</title></head> \n”);
                    “<Body bgcolor = \”#F4F5E3\” > \n <UL> \n” +
                    “<LI> User Name:” +
                    request.getParameter(“userName”) + “ \n” +
                    “<LI> Department:” +
                    request.getParameter(“deptName”) + “ \n” +
                    “</UL> \n </Body> \ n <html>”);
 }
}
```

Corresponding JSP

```
<html>
      <head>
            <title>Reading Request Parameters </title>
      </head>
 <Body bgcolor = #F4F5E3 >
   <UL>
      <LI> User Name: <%= request.getParameter(“userName”) %>
      <LI> Department: <%= request.getParameter(“deptName”) %>
   </UL>
 </Body>
```

```
<html>
```

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

<% java source code %>

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
        <% out.print("welcome to jsp");
%> </body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
    <body>
        <form action="welcome.jsp">
          <input type="text" name="uname">
          <input type="submit" value="go"><br/>
        </form>
   </body>
</html>
```

File: welcome.jsp

```
<html>
  <body>
        <form>
```

```
        <%
            String name=request.getParameter("uname");
            out.print("welcome "+name);
        %>
    </form>
    </body>
</html>
```

JSP Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage of Include directive:- Code Reusability
Syntax of include directive
<%@ include file="resourceName" %>

Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>
    <%@ include file="Header.html" %>
    Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

<%@ include file=”filename” %> is the JSP include directive.

At JSP page translation time, the content of the file given in the include directive is ‘pasted’ as it is, in the place where the JSP include directive is used. Then the source JSP page is converted into a java Servlet class. The included file can be a static resource or a JSP page. Generally JSP include directive is used to include header banners and footers.

The JSP compilation procedure is that, the source JSP page gets compiled only if that page has changed. If there is a change in the included JSP file, the source JSP file will not be compiled and therefore the modification will not get reflected in the output.

<jsp:include page=”relativeURL” /> is the JSP include action element.

When the included JSP page is called, both the request and response objects are passed as parameters.

If there is a need to pass additional parameters, then <jsp:param> element can be used. If the resource is static, its content is inserted into the calling JSP file, since there is no processing needed.

Example: Passing Parameter in <jsp:include>

```
<html> <head> <title>JSP include with parameters</title></head>
      <body>
            <jsp:include page="InstituteList.jsp" >
                  <jsp:param name="governingBody" value="Government" />
                  <jsp:param name="instituteType" value="Polytechnic" />
                  <jsp:param name="insttituteLocation" value="Lohaghat" />
            </jsp:include>
      </body>
</html>
```

JSP Action Tags Description

1.	jsp:forward	Forwards the request & response to other resource.
2.	jsp:include	Includes another resource.
3.	jsp:useBean	Creates or locates bean object.
4.	jsp:setProperty	Sets the value of property in bean object.
5.	jsp:getProperty	Prints the value of property of the bean.
6.	jsp:plugin	Embeds another components such as applet.
7.	jsp:param	Sets the parameter values used in forward and include.

Java Beans

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications. Following are the unique characteristics that distinguish a JavaBean from other Java classes –

1. It provides a default, no-argument constructor.
2. It should be serializable and that which can implement the Serializable interface.
3. It may have a number of properties which can be read or written.
4. It may have a number of "getter" and "setter" methods for the properties.

The <jsp:useBean> action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created then it instantiates the bean.

Syntax of jsp:useBean action tag

<jsp:useBean id= "instanceName" scope= "page | request | session | application"

class= "packageName.className" type= "packageName.className" beanName="packageName.className | <%= expression >" > </jsp:useBean>

Attributes and Usage of <jsp:useBean> action tag

SN	Tags	Use
1	id	Used to identify the bean in the specified scope.
2	scope	Represents the scope of the bean. It may be page, request, session or application. The default scope is page.
3	page	Specifies that you can use this bean within the JSP page. The default scope is page.
4	Request	Specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
5	Session	Specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
6	application	Specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
7	Class	Instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
8	Type	Provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
9	beanName	Instantiates the bean using the instantiate() method of java.beans.Beans class

Example: Bean Class for User which is accessed by UserDetails.JSP

```
public class UserBean
{
        private String userName;
        private int userType;

        public Details() {}
```

```
        public String getUserName()
        {
                return userName;
        }
        public void setUserName(String userName)
        {
                this.userName = userName;
        }
        public int getUserType()
        {
                return userType;
        }
        public void setUserType (int userType)
        {
                this.age = userType;
        }
}
```

Example: JSP Page to send User Details to UserDetails.JSP

```
<html>
<head><title> User Form </title></head>
<body>
<form action="UserDetails.jsp" method="post">
        User Name : <input type="text" name="userName"><br>
        User Type : <input type="text" name="userType"><br>
                    <input type="submit" value="Register">
</form>
</body>
</html>
```

Example: JSP page to write and read Bean Properties (File UserDetails.JSP)

```
<html>
<head><title> Print User Details
</title></head> <body>
        <jsp:useBean  id="userInfo"  class="UserBean" />
        <jsp:setProperty property="*" name="userInfo"/>
        <h1> Received User Detail as Below</h1>
        User Name:<jsp:getProperty property="userName"
        name="userInfo"/> <br>
        User Type: <jsp:getProperty property="userType" name="userInfo"/>
</body>
</html>
```

Http Session

The servlet container uses HttpSession interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period,

across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using Cookies or URL Rewriting or Hidden Form Field or HttpSession Interface.

HttpSession interface allows servlets to View and manipulate information about a session, such as the session identifier, creation time, and last accessed time Bind objects to sessions, allowing user information to persist across multiple user connections.

```
HttpSession session = request.getSession(true);
```

true : Use existing session if exist or create one new session
false : Use existing session if exist or return null

```
HttpSession session = request.getSession();
                          //Same as request.getSession(true);
```

Setting inactive time and values in session

```
session.setAttribute("user", username);
session.setMaxInactiveInterval(30);
```

Destroy Session in JSP and Servlet

1- There are some way to prevent session in JSP. It will not create session for this page.

```
<%@ page session="false"%>
```

2- Use session object to destroy the session
sesssion.invalidate();

3- Set Time till session is valid. If no session is active no session is created and it expires immediately

```
HttpSession session = request.getSession (false);

if (session != null)
        session.setMaxInactiveInterval(1);
```

Example: HTML Page Login.html Which Calls Servlet That Creates Session.

```
<html><head><title>Insert title
here</title></head> <body>
    <h1> Http Session HTML Page </h1>
    <form action="LoginController" method="post">
        User name : <input type="text" name="userName"><br>
        Password : <input type="password" name="password"><br>
        <input type="submit" value="Login">
    </form>
</body>
</html>
```

Example: Servlet Which Creates Session.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginController extends HttpServlet
{
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
                              throws ServletException, IOException
    {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        String uname = request.getParameter("userName");
        String pwd = request.getParameter("password");

        if (un.equals("GP Lohaghat"))

        {
            out.print("Welcome, " + uname);
            HttpSession session = request.getSession(true);
            session.setAttribute("userName", uname);
```

```
                session.setMaxInactiveInterval(60);
                response.sendRedirect("Home.jsp");
            }
            else
            {
                RequestDispatcher rd = null;
                rd = request.getRequestDispatcher ("Login.html");
                out.println("<font color=red>Wrong
                Credentials</font>"); rd.include(request, response);
            }
        }
}
```

Example: Home.JSP Which Reads Session Attributes.

```
<html>
<head><title>Accessing Session Attribute</title></head>
<body>

    <%
        String name = null;
        if (session != null)
        {
            if (session.getAttribute("user") != null) {
                name = (String) session.getAttribute("userName");
                out.print("Hello, " + name + "Welcome");
            }
            else
                response.sendRedirect("login.html");
        }
    %>


    <form action="LogoutController" method = "post">
        <input type="submit"
    value="Logout"> </form>
</body>
</html>
```

Example: Servlet Which Destroy Session.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LogoutController extends HttpServlet
{
	protected void doPost(HttpServletRequest request,
				HttpServletResponse response)
				Throws ServletException, IOException
	{
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
		out.println("Thank You. Session was destroyed successfully!!");
		HttpSession session = request.getSession(false);

		synchronized (session)
		{
			session.setAttribute("user", null);
			session.removeAttribute("user");
			session.getMaxInactiveInterval();
			session.invalidate();
		}
	}
}
```

MVC (Model View Controller)

MVC stands for Model View and Controller. It is a design pattern that separates the business logic and data access layers from the presentation layer.

Controller: Controller is a Servlet that acts as an interface between View and Model. Controller intercepts all the incoming requests.

Model: Model is a Java Bean which represents the state of the application i.e. data. It can also have business logic.

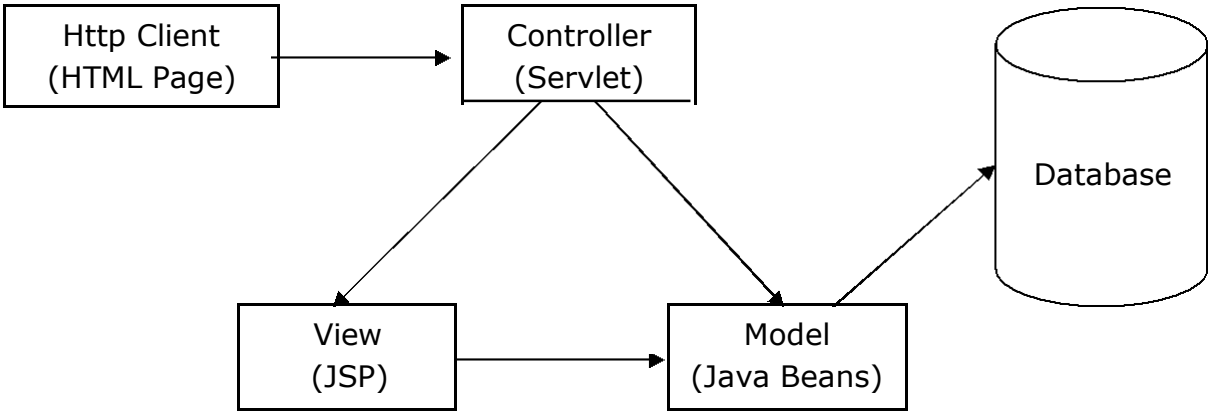
View: View is a JSP page which represents the presentation or User Interface.

Steps required to Implement MVC

1. Define beans to represent the data of application.
2. Use a Servlet to handle requests from clients.
6
3. Populate the beans by placing result obtained from accessing business logic and data access code in the beans.
4. Store the bean either in the request, session or Servlet context (application).

5. Forward the request to a JSP Page using forward method of RequestDispetcher interface or sendRedirect method of HttpServletResponse interface.

6. Extract data from the beans by accessing beans from JSP page using <jsp:useBean> and <jsp:getproperty> tags of JSP.



MVC Example in JSP

In this example, we are using Servlet as a controller, JSP as a view component, Java Bean class as a model.

In this example, we have created following pages:
index.jsp a page that gets input from the user.
ControllerServlet.java a Servlet that acts as a controller.
login-success.jsp and login-error.jsp files acts as view components.

Example: Login.html

```
<form action="ControllerServlet" method="post"> Name:<input
      type="text" name="name"><br> Password:<input
      type="password" name="password"><br> <input
      type="submit" value="login">
</form>
```

Example: Controller Servlet

```
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class ControllerServlet extends HttpServlet
{
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
                                        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        String name = request.getParameter("name"); String
        password = request.getParameter("password");

        LoginBean bean = new LoginBean();
        bean.setName(name);
        bean.setPassword(password);
        request.setAttribute("bean", bean);
        boolean status = bean.validate();

        RequestDispatcher rd = null;

        if(status) {
                rd = request.getRequestDispatcher("login-
                success.jsp"); rd.forward(request, response);
        }
        else {
                rd=request.getRequestDispatcher("login-
                error.jsp"); rd.forward(request, response);
        }
}



        protected void doGet(HttpServletRequest req, HttpServletResponse
                                resp) throws ServletException, IOException
{
        doPost(req, resp);
}
}
```

File: LoginBean.java

```
public class LoginBean
{
```

```
	private String name, password;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public boolean validate()
	{
		if(password.equals("admin"))
			return true;
		else
			return false;
	}
}
```

File: login-success.jsp

```
<%@page import = "LoginBean" %>
<p>You are successfully logged in!</p>
<%
	LoginBean bean=(LoginBean)request.getAttribute("bean");
	out.print("Welcome, "+bean.getName());
%>
```

File: login-error.jsp

```
<p>Sorry! User Name or Password Error</p>
<%@ include file="Login.html" %>
```

Advantages of MVC

1. Faster and parallel development process.
2. MVC Application is device independent.
3. Easy to maintain the large application.

4. Support for asynchronous technique of development.
5. Modification does not affect the entire model.
6. MVC model returns the data without formatting.
7. MVC makes the overall code much easier to maintain, test, debug, and reuse.

Disadvantages of MVC

1. Increased complexity and Need multiple programmers.
2. Inefficiency of data access in view.
3. Difficulty of using MVC with modern user interface.
4. Knowledge on multiple technologies is required.