


## LOGIN.HTML:

```
<!--Changing The DOCTYPE-->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Welcome</title>
    <meta name="description" content="ShopCentre"/>
    <meta name="keywords" content="HTML,CSS,XML,JavaScript"/>
    <meta name="author" content="XYZ"/>
  </head>
  <body>
    <h1> SUBJECTS </h1>
    <table>
      <tr>
        <td>
          <form action="#action" method="post">
            Username: <input type="text" />
            Password: <input type="password"/>
            Locations<!--Changing the name to id-->
            <input type="checkbox" id="web" value="Shoes"/>
            Shoes
            <input type="checkbox" id="net"
            value="Watches"/> Watches
            Additional Description:<!--Changing the name to
            id-->
            <textarea rows = "5" cols = "50" id = "description">
            </textarea>
            <input type="submit" value="Submit"/>
            <input type="reset" value="Reset"/>
          </form>
        </td>
      </tr>
    </table>
    <map id="flower"><!--Changing the name to id-->
      <area shape="rect" coords="0,0,82,126" href="thing1.htm" alt="alt"/>
      <area shape="poly" coords="90,58,3" href="thing2.htm" alt="alt"/>
    </map>
  </body>
</html>
```

```
<area shape="circle" coords="124,58,8" href="thing3.htm" alt="alt"/>
</map>
Contact:
<a href="http://www.ssn.edu.in/">SSN COLLEGE OF ENGINEERING</a>
</body>
</html>
```


## OUTPUT:

 **Markup Validation Service**  
Check the markup (HTML, XHTML, ...) of Web documents

**Jump To:** [Notes and Potential Issues](#) [Congratulations · Icons](#)

**This document was *Tentatively* checked as XHTML 1.0 Transitional**


<b>Result:</b>	Tentatively passed, <b>2 warning(s)</b>		
<b>File:</b>	<input type="button" value="Choose file"/> No file chosen <small>Use the file selection box above if you wish to re-validate the uploaded file xhtml.html</small>		
<b>Encoding:</b>	utf-8	<input type="button" value="(detect automatically)"/>	
<b>Doctype:</b>	XHTML 1.0 Transitional	<input type="button" value="(detect automatically)"/>	
<b>Root Element:</b>	html		
<b>Root Namespace:</b>	<a href="http://www.w3.org/1999/xhtml">http://www.w3.org/1999/xhtml</a>		



Interested in "developing" your developer skills? In W3C's hands-on Professional Certificate Program, learn how to code the right way by creating Web sites and apps that use the latest Web standards. [Find out more!](#)


[Donate](#) and help us build better tools for a better web.


## OUTPUT:

 ShoppingCentre.in


OffersShoesWatchesElectronics

LoginRegisterHi, SHADYLog Out






[Shoes](#)




Nike Shoe1  
₹ 1258 (10% OFF)

Add to Cart!




Nike Shoe2  
₹ 1257 (10% OFF)

Add to Cart!




Adidas Shoe1  
₹ 1350 (10% OFF)

Add to Cart!




Adidas Shoe2  
₹ 3750 (20% OFF)

Add to Cart!



Casual  
₹ 1450

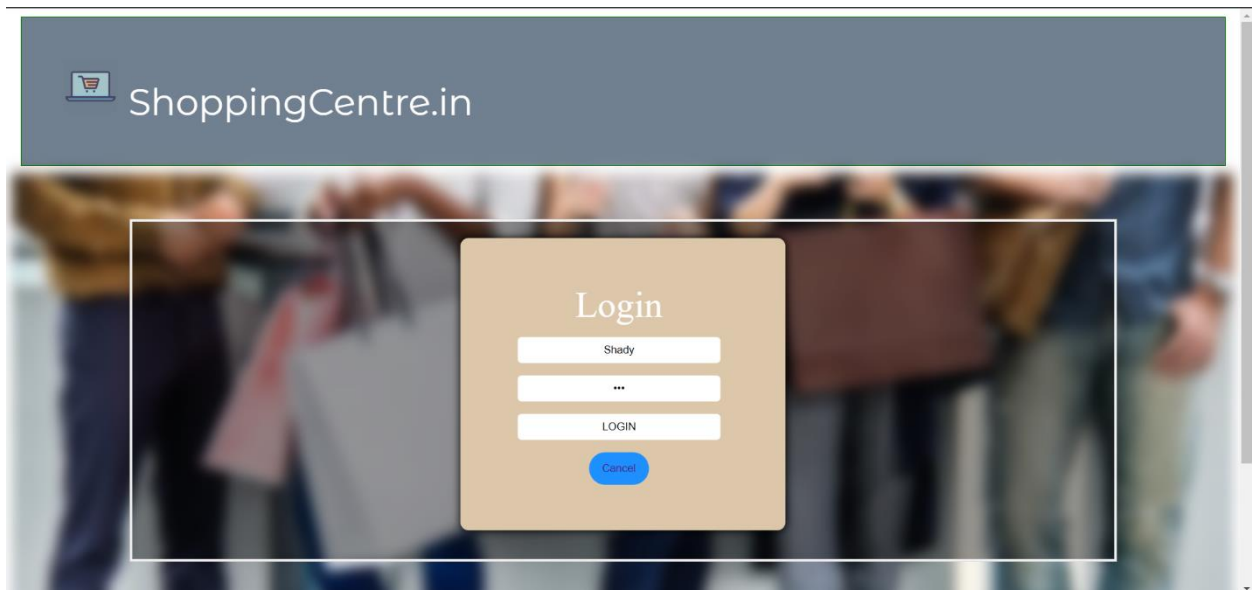
Add to Cart!



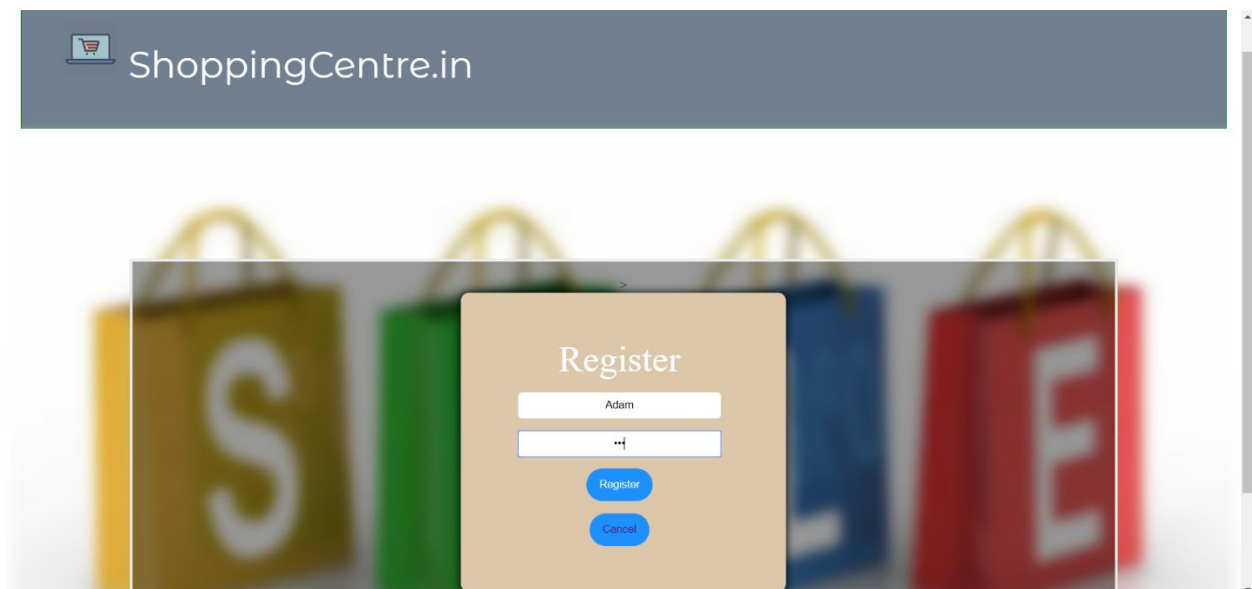
Leather  
₹ 5500 (15% OFF)

Add to Cart!

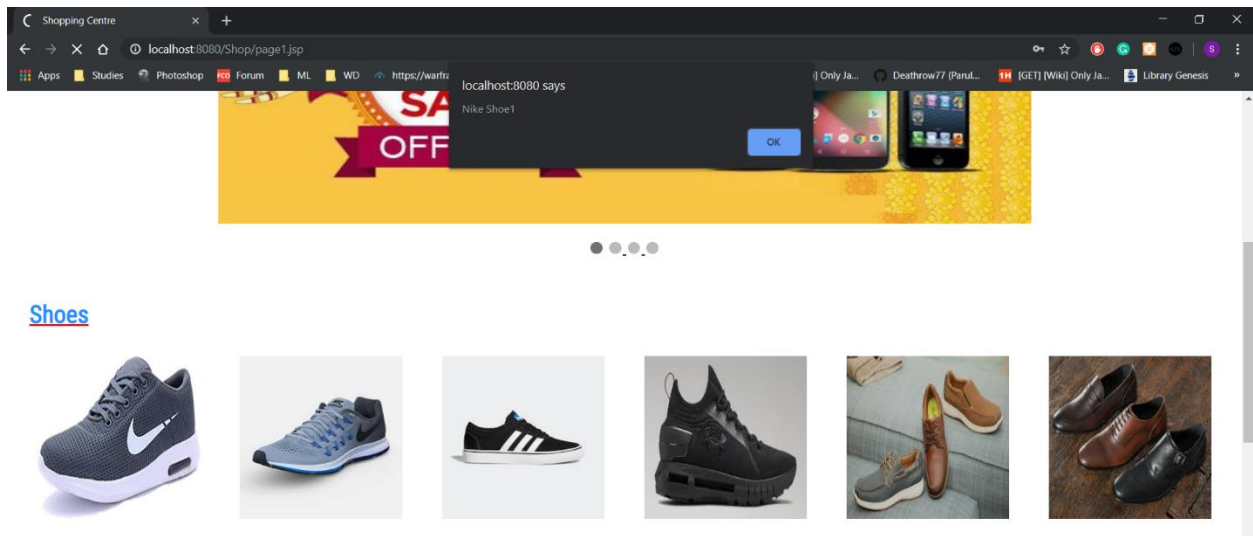
## LOGIN:



## REGISTER:



## OUTPUT:



**EX. NO: 6**

**DATE:**

## **JAVA SERVLETS**

Servlets are Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, then send response back to the web server.

### **Properties of Servlets :**

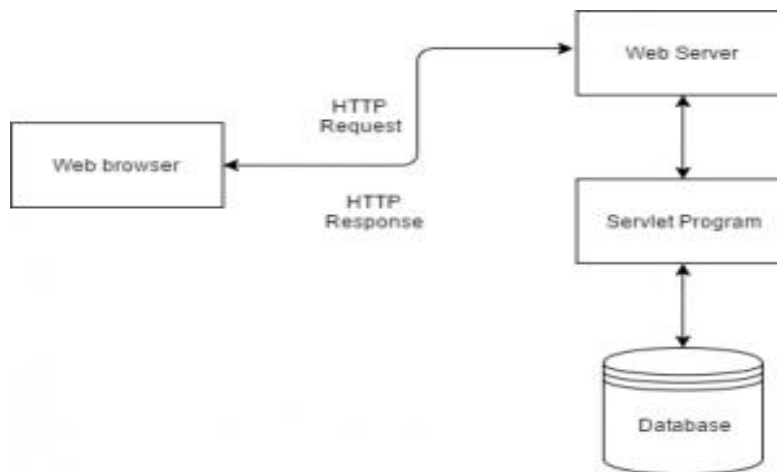
- Servlets work on the server-side.
- Servlets capable of handling complex request obtained from web server.

### **Execution of Servlets :**

Execution of Servlets involves the six basic steps:

1. The clients send the request to the web server.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generate the response in the form of output.
5. The servlet sends the response back to the web server.
6. The web server sends the response back to the client and the client browser displays it on the screen.

### **Servlet Architecture:**



### **Servlet container:**

Servlet container, also known as Servlet engine is an integrated set of objects that provide run time environment for Java Servlet components.

In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

Services provided by the Servlet container :

- Network Services : Loads a Servlet class. The loading may be from a local file system, a remote file system or other network services. The Servlet container provides the network services over which the request and response are sent.
- Decode and Encode MIME based messages : Provides the service of decoding and encoding MIME-based messages.
- Manage Servlet container : Manages the lifecycle of a Servlet.
- Resource management : Manages the static and dynamic resource, such as HTML files, Servlets and JSP pages.
- Security Service : Handles authorization and authentication of resource access.
- Session Management : Maintains a session by appending a session ID to the URL path.

## **Java JDBC:**

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

### **Basic steps to use a database in Java:**

- 1.Establish a connection
- 2.Create JDBC Statements
- 3.Execute SQL Statements
- 4.GET ResultSet
- 5.Close connections

## **CODE:**

### **LOGIN CHECK.JAVA:**

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

public class Login extends HttpServlet{
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)throws ServletException, IOException{
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        try{
            Class.forName("com.mysql.jdbc.Driver");
```

```

        Connection con = DriverManager.getConnection("jdbc:mysql:/
/localhost:3306/form","root","sudhakar");

        String uname = request.getParameter("uname");
        String psw = request.getParameter("psw");
        HttpSession session=request.getSession(true);

        PreparedStatement stmt = con.prepareStatement("select unam
e,pass from login where uname = ? and pass = ?");
        stmt.setString(1,uname);
        stmt.setString(2,psw);
        ResultSet rs = stmt.executeQuery();

        String j = request.getParameter("uname");
        request.setAttribute("user", j);
        if(rs.next()){
            RequestDispatcher rd = request.getRequestDispatcher
r("page1.jsp");
            rd.forward(request,response);
            return;
        }
        else{
            out.print("No user found");}

    }
    catch(Exception e){out.print(e);}
}
}

```

### **REGISTER.JAVA:**

```

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

public class Register extends HttpServlet{
    protected void service(HttpServletRequest request, HttpServletResponse
onse response) throws ServletException, IOException {
        doPost(request, response);
    }

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

```



```

        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql:/
localhost:3306/form","root","sudhakar");

        String uname = request.getParameter("uname");
        String psw = request.getParameter("psw");

        PreparedStatement pst = con.prepareStatement("inse
rt into login values(?,?)");
        pst.setString(1,uname);
        pst.setString(2,psw);
        pst.executeUpdate();
        out.print("Successs");

    }catch(Exception e){
        out.print(e);
    }
}
}

```

#### **ADDTOCART. JAVA**

```

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

public class Addtocart extends HttpServlet{
    protected void service(HttpServletRequest request, HttpServletResponse
onse response) throws ServletException, IOException {
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException{
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        try{
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql:/
localhost:3306/form","root","sudhakar");
            HttpSession session=request.getSession(false);
            if(session==null){
                out.print("<script>");
                out.print("alert('Please Login First');");
            }
        }
    }
}

```

```

        out.print("</script>");
        RequestDispatcher rd = request.getRequestDispatcher("login.html");
        rd.forward(request, response);
    }
    else{
        String uname=(String)session.getAttribute("name");
        String order = request.getParameter("order");
        String ordercost = request.getParameter("ordercost");
        session.setAttribute("order", order);
        session.setAttribute("ordercost", ordercost);
        PreparedStatement stmt = con.prepareStatement("insert
into orders values (?, ?, ?)");
        stmt.setString(1, uname);
        stmt.setString(2, order);
        stmt.setString(3, ordercost);
        stmt.execute();
        RequestDispatcher rd = request.getRequestDispatcher("input.jsp");
        rd.forward(request, response);
        return;
    }
    out.close();
} catch (Exception e)
{
    out.print(e);
}
}}

```

#### **LOGOUT. JAVA**

```

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

        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("page1.jsp").forward(request, response);

        HttpSession session=request.getSession();
        session.invalidate();
    }
}

```

```

        out.print("You are successfully logged out!");
    out.close();
} }

```

## OUTPUT:

The image displays the output of a web application, showing two main forms: the Login form and the Register form, both overlaid on a background image of shopping bags.

**Login Form:** The form is titled "Login" and is centered on a light brown background. It contains two input fields: the first is labeled "Shady" and the second is labeled "...". Below these fields is a "LOGIN" button and a "Cancel" button.

**Register Form:** The form is titled "Register" and is centered on a light brown background. It contains two input fields: the first is labeled "Adam" and the second is labeled "...". Below these fields is a "Register" button and a "Cancel" button.

**MySQL Workbench Screenshot:** The screenshot shows the MySQL Workbench interface. The left sidebar displays the "SCHEMAS" tree, with the "form" schema selected. The "form" schema contains tables: "login", "register", and "cart". The "login" table is selected, and its columns are listed: "uname", "pass", "order", "ordercost", and "uname". The "register" table is also listed with columns: "uname", "pass", "order", "ordercost", and "uname". The "cart" table is listed with columns: "uname", "pass", "order", "ordercost", and "uname". The main query editor shows a query: `SELECT * FROM form.login;`. The "Result Grid" at the bottom displays the results of the query, showing a table with columns "uname" and "pass". The results are as follows:

uname	pass
Shady	123
sss	sss
ssss	ssss
Sudhakar	shady
sudhakarj	sudhakar
Viewesh	123
12333	12333

**EX. NO: 7**

**DATE:**

### **MVC APPLICATION USING JSP, BEAN AND DATABASE**

One of the most common Design Patterns is Model-View-Controller (MVC). MVC stands for Model, View and Controller. MVC separates application into three components - Model, View and Controller.

**Model:** Model represents shape of the data and business logic. It maintains the data of the application. Model objects retrieve and store model state in a database.

Model is a data and business logic.

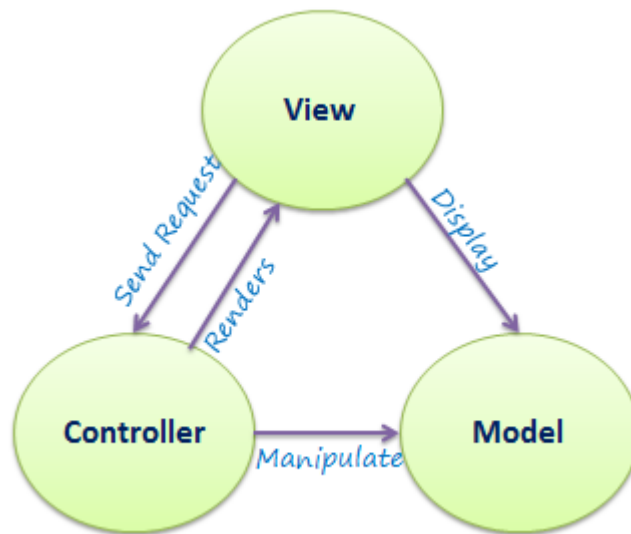
**View:** View is a user interface. View display data using model to the user and also enables them to modify the data.

View is a User Interface.

**Controller:** Controller handles the user request. Typically, user interact with View, which in-turn raises appropriate URL request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response.

Controller is a request handler.

## MVC ARCHITECTURE:



### Advantages of MVC:

One advantage is *separation of concerns*

Computation is not intermixed with I/O

Consequently, code is cleaner and easier to understand

Another advantage is flexibility

The GUI (if one is used) can be completely revamped without touching the model in any way

Another big advantage is reusability

The same model used for a servlet can equally well be used for an application or an applet (or by another process)

### CODE:

#### hello.jsp:

```
<%@ page language="java" contentType="text/html"
pageEncoding="UTF-8"%>
```

```

<%@ page import="
java.io.*,javax.servlet.*,javax.servlet.http.*" %>

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

<html lang="en" dir="ltr">

    <head>

        <meta charset="utf-8">

        <title></title>

    </head>

    <body>

<%

        response.setContentType("text/html");

        String uid = (String)session.getAttribute("uname");

            if (uid == null)

                {

%>

                <script type="text/javascript">

                    window.onload=function(){

                        alert("Please Login!");

                    }

                </script>

                    <jsp:include page="index.jsp"/>

<%

                }

                else

                {

                    %>

                    <jsp:useBean id="catinfo" class="bean.validatebean">
</jsp:useBean>

```

```

        <jsp:setProperty property="*" name="logininfo"/>
        <jsp:getProperty property="uname" name="logininfo"/>
        <jsp:forward page="details.jsp">
            <jsp:param name="cat" value="<%= logininfo.getuname()
%>" />

        </jsp:forward>

        hello,<%= logininfo.getuname() %>

            <% } %>

    </body>
</html>

```

### **CheckBean.java:**

```

package bean;

public class validatebean
{
    String uname;

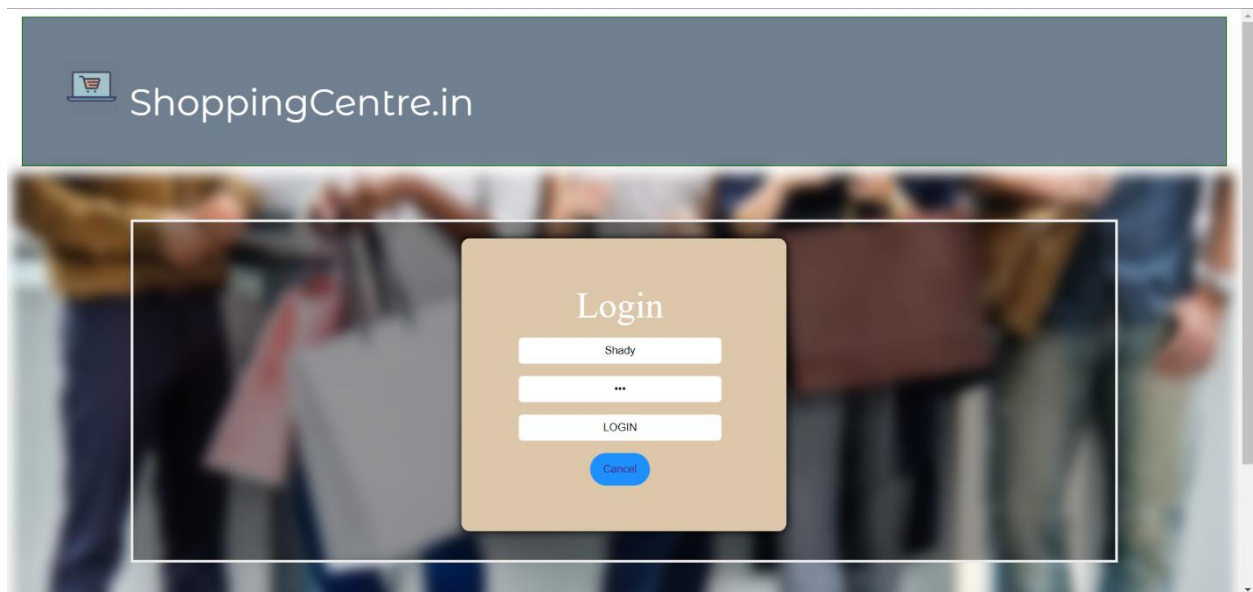
    public validatebean( ) {    }

    public void setuname(String uname)
    {
        this.uname = uname;
    }

    public String getuname( )
    {
        return uname;
    }
}

```

## OUTPUT:





**EX. NO: 8**

**DATE:**

## **XML VALIDATION**

### **XML (Extensible Markup Language)**

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the W3C's XML 1.0 Specification and by several other related specifications, all of which are free open standards.

The design goals of XML emphasize simplicity, generality and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures such as those used in web services.

### **XML Declaration:**

XML documents may begin by declaring some information about themselves, as in the following example:

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

### **Well-Formed XML Document:**

The XML specification defines an XML document as a well-formed text – meaning that it satisfies a list of syntax rules provided in the specification. Some key points in the fairly lengthy list include:

- The document contains only properly encoded legal Unicode characters
- None of the special syntax characters such as < and & appear except when performing their markup-delineation roles
- The begin, end, and empty-element tags that delimit the elements are correctly nested, with none missing and none overlapping
- The element tags are case-sensitive; the beginning and end tags must match exactly.
- Tag names cannot contain any of the characters !"#\$%&'()\*+,-/;<=>?@[\\]^`{|}~, nor a space character, and cannot start with -, ., or a numeric digit.
- A single "root" element contains all the other elements.

The definition of an XML document excludes texts that contain violations of well-formedness rules; they are simply not XML. An XML processor that encounters such a violation is required to report such errors and to cease normal processing. This policy, occasionally referred to as "draconian error handling," stands in notable contrast to the behavior of programs that process HTML, which are designed to produce a reasonable result even in the presence of severe markup errors. XML's policy in this area has been criticized as a violation of Postel's law ("Be conservative in what you send; be liberal in what you accept").

The XML specification defines a valid XML document as a well-formed XML document which also conforms to the rules of a Document Type Definition (DTD).

### **Valid XML Document:**

In addition to being well-formed, an XML document may be valid. This means that it contains a reference to a Document Type Definition (DTD) and that its elements and attributes are declared in that DTD and follow the grammatical rules for them that the DTD specifies.

XML processors are classified as validating or non-validating depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue normal processing.

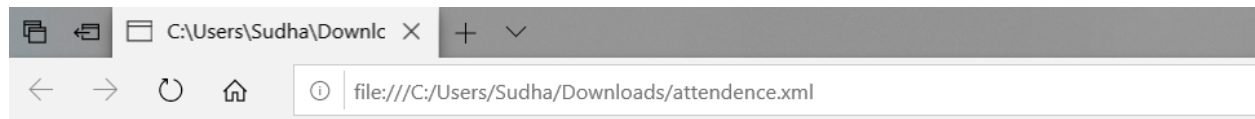
A DTD is an example of a schema or grammar. Since the initial publication of XML 1.0, there has been substantial work in the area of schema languages for XML. Such schema languages typically constrain the set of elements that may be used in a document, which attributes may be applied to them, the order in which they may appear, and the allowable parent/child relationships.

### **CODE:**

```
<?xml version = "1.0" ?>

<person>
    <name> Sudhakar </name>
    <email> Sudhakar.jeeva7@gmail.com </email>
    <mobile> 9080663323 </mobile>
    <gender>
        <male> Yes </male>
        <female> No </female>
    </gender>
    <place>
        <address>Anna nagar</address>
        <city>Chennai</city>
        <state>Tamilnadu</state>
    </place>
</person>
```

## OUTPUT:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <person>
  <name> Sudhakar </name>
  <email> Sudhakar.jeeva7@gmail.com </email>
  <mobile> 9080663323 </mobile>
  - <gender>
    <male> Yes </male>
    <female> No </female>
  </gender>
  - <place>
    <address> Anna nagar </address>
    <city> Chennai </city>
    <state> Tamilnadu </state>
  </place>
</person>
```

**EX. NO: 9**

**DATE:**

## **XSLT AND XML**

### **XSL:**

XSL stands for Extensible Stylesheet Language. CSS was designed for styling HTML pages, and can be used to style XML pages. XSL was designed specifically to style XML pages, and is much more sophisticated than CSS. XSL consists of three languages: XSLT (XSL Transformations) is a language used to transform XML documents into other kinds of documents (most commonly HTML, so they can be displayed). XPath is a language to select parts of an XML document to transform with XSLT. XSL-FO (XSL Formatting Objects) is a replacement for CSS

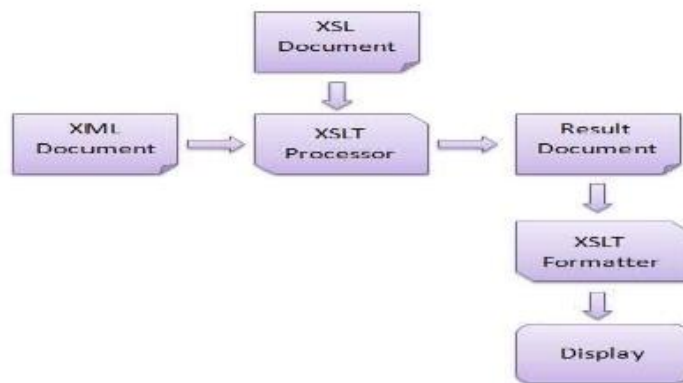
The XML source document is parsed into an XML source tree. You use XSLT to transform the matched part and put the transformed information into the result tree. The result tree is output as a result document. Parts of the source document that are not matched by a template are typically copied unchanged

### **Need for XSL**

In case of HTML document, tags are predefined such as table, div, and span; and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Stylesheet Language. An XSL document specifies how a browser should render an XML document.

### **How XSLT Works**

An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document. XSLT stylesheet is written in XML format. XSLT Processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.



## Advantages

Here are the advantages of using XSLT –

- Independent of programming. Transformations are written in a separate xsl file which is again an XML document.
- Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

## XML CODE:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="planes.xsl"?>
<planes_for_sale>
  <ad>
    <model>SkyHawk</model>
    <price>5349500</price>
  </ad>
  <ad>
    <model>Jet</model>
    <price>1949500</price>
  </ad>
</planes_for_sale>
```

## XSLT CODE:

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

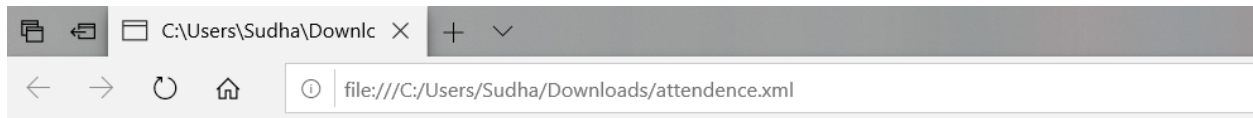
```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>Planes</h2>
<table>
  <tr>
    <th>Model</th>
    <th>Price</th>
    <th>Class</th>
  </tr>
  <xsl:for-each select="planes_for_sale/ad">
    <tr>
      <td>
        <xsl:value-of select="model" />
      </td>
      <td>
        <xsl:value-of select="price" />
      </td>
      <td>
        <xsl:choose>
          <xsl:when test="price < 2000000">
Eco
          </xsl:when>
          <xsl:when test="price > 2000000 and price <
5000000">
Mid
          </xsl:when>
          <xsl:otherwise>
Lux
          </xsl:otherwise>
        </xsl:choose>
      </td>
    </tr>
  </xsl:for-each>
</table>
</body>

```

```
</html>
</xsl:template>
</xsl:stylesheet>
```

## OUTPUT:



### IT Attendance

Regno	Student	dob	class	year	cgpa
114	Sudhakar	1999	ITB	3	7.5
126	Viswesh	1998	ITB	3	8.9
115	Sudharshan	1999	ITB	3	9.90
123	Veeramani	1990	ITA	3	10.20

**EX. NO: 10**

**DATE:**

### **XML: Document Object Model (DOM)**

The XML Document Object Model (DOM) class is an in-memory representation of an XML document. The DOM allows you to programmatically read, manipulate, and modify an XML document. Editing is the primary function of the DOM. It is the common and structured way that XML data is represented in memory, although the actual XML data is stored in a linear fashion when in a file or coming in from another object.

The XML DOM defines a standard way for accessing and manipulating XML documents.

The XML DOM views an XML document as a tree-structure.

All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.

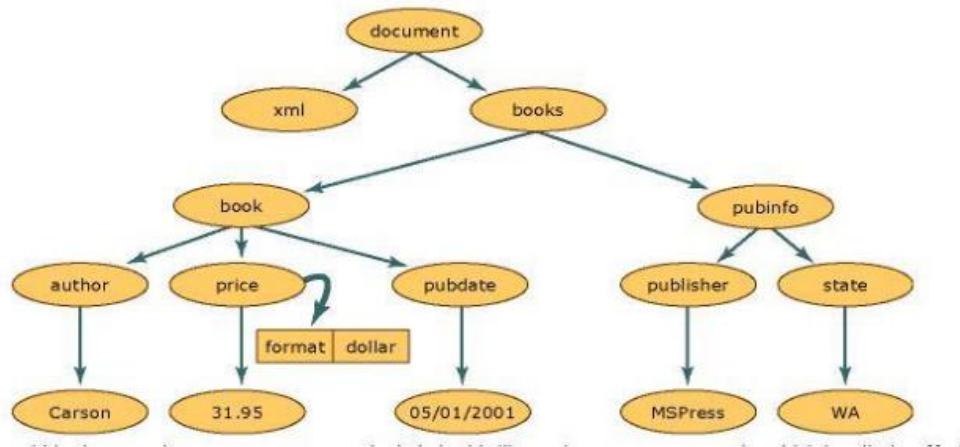
#### **XML INPUT:**

```
<?xml version="1.0"?>
<books>
  <book>
    <author>Carson</author>
    <price format="dollar">31.95</price>
    <pubdate>05/01/2001</pubdate>
  </book>
  <pubinfo>
    <publisher>MSPress</publisher>
    <state>WA</state>
  </pubinfo>
</books>
```

The following illustration shows how memory is structured when this XML data is read into the DOM structure.



## XML document structure:



## XML CODE:

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

```
<hospital_data>
```

```
  <hospital>
```

```
    <details>
```

```
      <id>241117</id>
```

```
      <name>Apollo</name>
```

```
      <location>Chennai</location>
```

```
    </details>
```

```
    <patient>
```

```
      <doc_id>D55</doc_id>
```

```
      <pid>111234</pid>
```

```
      <dob>29-sep-1997</dob>
```

```
      <illness>Asthma</illness>
```

```
      <doj>20-jun-2018</doj>
```

```
    </patient>
```

```
  </hospital>
```

```

        <doc_id>D89</doc_id>

        <pid>111987</pid>

        <dob>10-jan-1999</dob>

        <illness>Wheezing</illness>

        <doj>29-aug-2019</doj>

    </patient>

    <doctor>

        <doc_id>D55</doc_id>

        <dname>xyz</dname>

        <expert>immunologist</expert>

    </doctor>

    <doctor>

        <doc_id>D89</doc_id>

        <dname>abc</dname>

        <expert>Neck</expert>

    </doctor>

</hospital>

</hospital_data>

```

### **JAVA CODE:**

```

import java.io.File;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

public class DomHospxml {

```

```

    public static void main(String[] args){
        try {
            File fXmlFile = new File("hosp.xml");
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            // Document doc = dBuilder.parse(fXMLFile);
            Document doc = dBuilder.parse(fXmlFile);
            doc.getDocumentElement().normalize();

            System.out.println("Root Element : " +
doc.getDocumentElement().getNodeName());

            NodeList nList1 = doc.getElementsByTagName("patient");
            NodeList nList2 = doc.getElementsByTagName("doctor");

            System.out.println("-----");

            for(int temp = 0; temp < nList1.getLength(); temp++){
                Node nNode = nList1.item(temp);

                System.out.println("\nCurrent Element : " +
nNode.getNodeName());

                if(nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;

                    System.out.println("Doctor ID : " +
eElement.getElementsByTagName("doc_id").item(0).getTextContent());
                    System.out.println("Patient ID : " +
eElement.getElementsByTagName("pid").item(0).getTextContent());
                    System.out.println("DateOfBirth : " +
eElement.getElementsByTagName("dob").item(0).getTextContent());
                    System.out.println("Sickness : " +
eElement.getElementsByTagName("illness").item(0).getTextContent());
                    System.out.println("DateOfJoining : " +
eElement.getElementsByTagName("doj").item(0).getTextContent());
                }
            }
            System.out.println("-----");
        }
    }

```

```

        for(int i = 0; i < nList2.getLength(); i++){
            Node nNode2 = nList2.item(i);

            System.out.println("\nCurrent Element : " +
nNode2.getNodeName());

            if(nNode2.getNodeType() == Node.ELEMENT_NODE) {
                Element elt = (Element) nNode2;

                System.out.println("DOCTOR ID : " +
elt.getElementsByTagName("doc_id").item(0).getTextContent());
                System.out.println("DOCTOR NAME : " +
elt.getElementsByTagName("dname").item(0).getTextContent());
                System.out.println("SPECAILIST : " +
elt.getElementsByTagName("expert").item(0).getTextContent());
            }
        }
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
}

```

## OUTPUT:

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\sudharshan B>cd Desktop/web/xml related/xml parser

C:\Users\sudharshan B\Desktop\web\xml related\xml parser>javac DomHospxml.java

C:\Users\sudharshan B\Desktop\web\xml related\xml parser>java DomHospxml
Root Element : hospital_data
-----

Current Element : patient
Doctor ID      : D55
Patient ID     : 111234
DateOfBirth    : 29-sep-1997
Sickness       : Asthma
DateOfJoining  : 20-jun-2018

Current Element : patient
Doctor ID      : D89
Patient ID     : 111987
DateOfBirth    : 10-jan-1999
Sickness       : Wheezing
DateOfJoining  : 29-aug-2019
-----

Current Element : doctor
DOCTOR ID      : D55
DOCTOR NAME    : xyz
SPECAILIST     : immunologist

Current Element : doctor
DOCTOR ID      : D89
DOCTOR NAME    : abc
SPECAILIST     : Neck

C:\Users\sudharshan B\Desktop\web\xml related\xml parser>
```

**EX. NO: 11**

**DATE:**

### **XML: SIMPLE API FOR XML (SAX)**

SAX (Simple API for XML) is an event-driven online algorithm for parsing XML documents, with an API interface developed by the XML-DEV mailing list. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

#### **BENEFITS:**

SAX parsers have some benefits over DOM-style parsers. A SAX parser only needs to report each parsing event as it happens, and normally discards almost all of that information once reported (it does, however, keep some things, for example, a list of all elements that have not been closed yet, in order to catch later errors such as end-tags in the wrong order). Thus, the minimum memory required for a SAX parser is proportional to the maximum depth of the XML file (i.e., of the XML tree) and the maximum data involved in a single XML event (such as the name and attributes of a single start-tag, or the content of a processing instruction, etc.).

This much memory is usually considered negligible. A DOM parser, in contrast, typically builds a tree representation of the entire document in memory, to begin with, thus using memory that increases with the entire document length. This takes considerable time and space for large documents (memory allocation and data-structure construction take time). The compensating advantage, of course, is that once loaded any part of the document can be accessed in any order.

Because of the event-driven nature of SAX, processing documents is generally far faster than DOM-style parsers, so long as the processing can be done in a start-to-end pass. Many tasks, such as indexing, conversion to other formats, very simple formatting, and the like, can be done that way. Other tasks, such as sorting, rearranging sections, getting from a link to its target, looking up information on one element to help process a later one, and the like, require accessing the document structure in complex orders and will be much faster with DOM than with multiple SAX passes.

#### **EXAMPLE:**

```
<?xml version="1.0" encoding="UTF-8"?>
<DocumentElement param="value">
  <FirstElement>Some Text
</FirstElement>
  <?some_pi some_attr="some_value"?>
  <SecondElement param2="something">
    Pre-Text <Inline>Inlined text</Inline> Post-text.
  </SecondElement>
</DocumentElement>
```

This XML document, when passed through a SAX parser, will generate a sequence of events like the following:

- ❖ XML Element start, named DocumentElement, with an attribute param equal to "value"
- ❖ XML Element start, named FirstElement
- ❖ Some Text" (note: certain white spaces can be changed)
- ❖ XML Element end, named FirstElement
- ❖ XML Text node, with data equal to "Pre-Text"
- ❖ XML Element start, named Inline
- ❖ XML Text node, with data equal to "Inlined text"
- ❖ XML Element end, named Inline
- ❖ XML Text node, with data equal to "Post-text."
- ❖ XML Element end, named SecondElement
- ❖ XML Element end, named DocumentElement

### **XML CODE:**

```
<?xml version = "1.0" ?>
<person>
    <name> Sudhakar </name>
    <email> Sudhakar.jeeva7@gmail.com </email>
    <mobile> 9080663323 </mobile>
    <gender>
        <male> Yes </male>
        <female> No </female>
    </gender>
    <place>
        <address>Anna nagar</address>
        <city>Chennai</city>
        <state>Tamilnadu</state>
    </place>
</person>
```

### **JAVA CODE:**

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
public class sax{
    public static void main(String[] args) throws IOException{
        sax detail = new sax("reg1.xml");
```

```

    }
    public sax(String str){
        try{
            File file = new File(str);
            if (file.exists()){
                SAXParserFactory parserFact =
SAXParserFactory.newInstance();
                SAXParser parser = parserFact.newSAXParser();
                System.out.println("XML Data: ");
                DefaultHandler dHandler = new
DefaultHandler(){
                    boolean n;
                    boolean em;
                    boolean mob;
                    public void startElement(String uri, String
localName, String element_name, Attributes attributes)throws
SAXException{

                        if (element_name.equals("name")){
                            n = true;
                        }
                        if (element_name.equals("email")){
                            em = true;
                        }
                        if (element_name.equals("mobile")){
                            mob = true;
                        }
                    }

                }

                public void characters(char[] ch, int start,
int len) throws SAXException{
                    String str = new String (ch, start, len);

                    if (n){
                        System.out.println("Name: "+str);
                        n = false;
                    }
                    if (em){

```



```

        System.out.println("Email: "+str);
        em = false;
    }
    if (mob){
        System.out.println("Mobile: "+str);
        mob = false;
    }

    }

    };
    parser.parse(str, dHandler);
}
else{
    System.out.println("File not found!");
}
}
catch (Exception e){
    System.out.println("XML      File      hasn't      any
elements");
    e.printStackTrace();
}
}
}

```

## OUTPUT:

---

```

C:\Users\Hp\Downloads>javac sax.java

C:\Users\Hp\Downloads>java sax
XML Data:
Name:      Sameera Holy
Email:     sameeraholy@gmail.com
Mobile:    8220658042

C:\Users\Hp\Downloads>

```

**EX. NO:12**

**DATE:**

## **XML VALIDATING DTD AND XML-SCHEMA USING PARSERS**

### **1) Document Type Definition(DTD):**

The oldest schema language for XML is the Document Type Definition (DTD), inherited from SGML.

DTDs have the following benefits:

- DTD support is ubiquitous due to its inclusion in the XML 1.0 standard.
- DTDs are terse compared to element-based schema languages and consequently present more information in a single screen.
- DTDs allow the declaration of standard public entity sets for publishing characters.
- DTDs define a document type rather than the types used by a namespace, thus grouping all constraints for a document in a single collection.

DTDs have the following limitations:

- They have no explicit support for newer features of XML, most importantly namespaces.
- They lack expressiveness. XML DTDs are simpler than SGML DTDs and there are certain structures that cannot be expressed with regular grammars. DTDs only support rudimentary datatypes.
- They lack readability. DTD designers typically make heavy use of parameter entities (which behave essentially as textual macros), which make it easier to define complex grammars, but at the expense of clarity.
- They use a syntax based on regular expression syntax, inherited from SGML, to describe the schema. Typical XML APIs such as SAX does not attempt to offer applications a structured representation of the syntax, so it is less accessible to programmers than an element-based syntax may be.

Two peculiar features that distinguish DTDs from other schema types are the syntactic support for embedding a DTD within XML documents and for defining entities, which are arbitrary fragments of text and/or markup that the XML processor inserts in the DTD itself and in the XML document wherever they are referenced as character escapes.

DTD technology is still used in many applications because of its ubiquity.

XML-DTD declaration:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

### XML CODE:

```
<?xml version="1.0"?>
<!DOCTYPE Register[
    <!ELEMENT register(name,email,mobile,gender,place) >
    <!ELEMENT name (#PCDATA) >
    <!ELEMENT email (#PCDATA) >
    <!ELEMENT mobile (#PCDATA) >
    <!ELEMENT gender(male,female) >
    <!ELEMENT male (#PCDATA) >
    <!ELEMENT female (#PCDATA) >
    <!ELEMENT place(address+,city+,state+) >
    <!ELEMENT address (#PCDATA) >
    <!ELEMENT city (#PCDATA) >
    <!ELEMENT state (#PCDATA) >
]>

<person>
    <name> Sudhakar </name>
    <email> Sudhakar.jeeva7@gmail.com </email>
    <mobile> 9080663323 </mobile>
    <gender>
        <male> Yes </male>
        <female> No </female>
    </gender>
    <place>
        <address>Anna nagar</address>
        <city>Chennai</city>
        <state>Tamilnadu</state>
    </place>
</person>
```

### OUTPUT:

## XML Validator Online BETA

### XML Validation

☐ XPath ☐ Schema ☒ DTD

Charset:

Choose XML file to validate:  regdtd.dtd

Due to the technical issue, embedded DTD is allowed ONLY.

### XPath Explorer

### Validation Result



Validation successful!

view plain print ?

```

1.  <?xml version="1.0"?>
2.  <!DOCTYPE Register[
3.      <!ELEMENT register (name,email,mobile,gender,place) >
4.      <!ELEMENT name (#PCDATA) >
5.      <!ELEMENT email (#PCDATA) >
6.      <!ELEMENT mobile (#PCDATA) >
7.      <!ELEMENT gender (male,female) >
8.      <!ELEMENT male (#PCDATA) >
9.      <!ELEMENT female (#PCDATA) >
10.     <!ELEMENT place (address+,city+,state+) >
11.     <!ELEMENT address (#PCDATA) >
12.     <!ELEMENT city (#PCDATA) >
13.     <!ELEMENT state (#PCDATA) >
14. ]>
15. <person>
16.     <name> Sudhakar </name>
17.     <email> Sudhakar.jeeva7@gmail.com </email>
18.     <mobile> 9080663323 </mobile>
19.     <gender>
20.         <male> Yes </male>
21.         <female> No </female>
22.     </gender>
23.     <place>
24.         <address>Anna nagar</address>
25.         <city>Chennai</city>
26.         <state>Tamilnadu</state>
27.     </place>
28. </person>

```

## SCHEMA IN XML

### XML SCHEMA:

A newer schema language, described by the W3C as the successor of DTDs, is XML Schema, often referred to by the initialism for XML Schema instances, XSD (XML Schema Definition). XSDs are far more powerful than DTDs in describing XML languages. They use a rich datatyping system and allow for more detailed constraints on an XML document's logical structure. XSDs also use an XML-based format, which makes it possible to use ordinary XML tools to help process them.

xs:schema element that defines a schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
</xs:schema>
```

### XML CODE:

```
<?xml version = "1.0" ?>
<person>
    <name> Sudhakar </name>
    <email> Sudhakar.jeeva7@gmail.com </email>
    <mobile> 9080663323 </mobile>
    <gender>
        <male> Yes </male>
        <female> No </female>
    </gender>
    <place>
        <address>Anna nagar</address>
        <city>Chennai</city>
        <state>Tamilnadu</state>
    </place>
</person>
```

### XSD CODE:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="name" type="xs:string"/>
<xs:element name="email" type="xs:string"/>
<xs:element name="mobile" type="xs:string"/>
<xs:element name="male" type="xs:string"/>
<xs:element name="female" type="xs:string"/>
```

```
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="state" type="xs:string"/>

<xs:element name="gender">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="male" />
      <xs:element ref="female" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="place">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="address" />
      <xs:element ref="city" />
      <xs:element ref="state" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" />
      <xs:element ref="email" />
      <xs:element ref="mobile" />
      <xs:element ref="gender" />
      <xs:element ref="place" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

## OUTPUT:

XML

```
<person>
  <name> Sudhakar </name>
  <email> Sudhakar.jeeva7@gmail.com </email>
  <mobile> 9080663323 </mobile>
  <gender>
    <male> Yes </male>
    <female> No </female>
  </gender>
  <place>
    <address>Anna nagar</address>
    <city>Chennai</city>
    <state>Tamilnadu</state>
  </place>
</person>
```

Check XML Well Formed

XSD Schema

```
<xs:element ref="address" />
<xs:element ref="city" />
<xs:element ref="state" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" />
      <xs:element ref="email" />
      <xs:element ref="mobile" />
      <xs:element ref="gender" />
      <xs:element ref="place" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Check XSD Validity

Validate XML against XSD

Result

The XML is Well Formed and Valid.

**EX. NO: 13**

**DATE:**

## **AJAX**

### **AJAX:**

AJAX stands for Asynchronous **J**avaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

### **Rich Internet Application Technology:**

AJAX is the most viable Rich Internet Application (RIA) technology so far. It is getting tremendous industry momentum and several tool kit and frameworks are emerging. But at the same time, AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.

### **AJAX is Based on Open Standards**

AJAX is based on the following open standards –

- Browser-based presentation using HTML and Cascading Style Sheets (CSS).
- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen.



**AJAX.js:**

```
function loadXMLDoc()
{
    var xmlhttp;
    var k=document.myform.uname.value;
    var url="checkusername.jsp?ver="+k;
    if (window.XMLHttpRequest)
    {
        xmlhttp=new XMLHttpRequest();
    }
    else
    {
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4)
        {
            document.getElementById("err").innerHTML=xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET",url,true);
    xmlhttp.send();
}
```

**Checkusername.jsp:**

```
<!-- <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%> --%>
<%@ page import="java.io.*,java.sql.*" %>
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<%
String sn=request.getParameter("ver");
Class.forName("com.mysql.jdbc.Driver");
Connection con
=DriverManager.getConnection("jdbc:mysql://localhost:3306/form",
"root", "sudhakar");
Statement st=con.createStatement();
ResultSet rs = st.executeQuery("select * from login where
name='"+sn+"'");
if(rs.next())
{
```

```
out.println("<font color=red>");  
out.println("&#10006;");  
out.println("</font>");  
}else {  
out.println("<font color=red>");  
out.println("&#10004;");  
out.println("</font>");  
}  
rs.close();  
st.close();  
con.close();  
%>
```

## OUTPUT:

