**SOFTWARE DEVELOPMENT TOOLS**

**HTML**

Html is a language which is used to create web pages with html marking up a page to indicate its format, telling the web browser where you want a new line to begin or how you want text or images aligned and more are possible.

We used the following tags in our project.

**Table:**

Tables are so popular with web page authors is that they let you arrange the elements of a web page in such a way that the browser won’t rearrange them web page authors frequently use tables to structure web pages.

**TR:**

TRis used to create a row in a table encloses <TH> and <TD> elements. <TR> contain many attributes. Some of them are,

* ALIGN: specifies the horizontal alignment of the text in the table row.
* BGCOLOR: Specifies the background color for the row.
* BORDERCOLOR: Sets the external border color for the row.
* VALIGN: Sets the vertical alignment of the data in this row.

**TH:**

TH is used to create table heading.

* ALIGN: Sets the horizontal alignment of the content in the table cell. Sets LEFT, RIGHT, CENTER.
* BACKGROUND: Species the back ground image for the table cell.
* BGCOLOR: Specifies the background color of the table cell
* VALIGN: Sets the vertical alignment of the data. Sets to TOP, MIDDLE, BOTTOM or BASELINE.
* WIDTH: Specifies the width of the cell. Set to a pixel width or a percentage of the display area.

**TD:**

TD is used to create table data that appears in the cells of a table.

* ALIGN: Species the horizontal alignment of content in the table cell. Sets to LEFT, CENTER, RIGHT.
* BGCOLOR: Specifies the background image for the table cell.
* BGCOLOR: sets the background color of the table cells.
* WIDTH: Species the width of the cell

**Frames:**

Frames are used for either run off the page or display only small slices of what are supposed to be shown and to configure the frame we can use <FRAMESET>There are two important points to consider when working with <FRAMESET>.

* <FRAMESET> element actually takes the place of the <BODY> element in a document.
* Specifying actual pixel dimensions for frames .

<FRAME> Elements are used to create actual frames.

From the frameset point of view dividing the browser into tow vertical frames means creating two columns using the <FRAMESET> elements COLS attribute.

The syntax for vertical fragmentation is,

<FRAMESET COLS =”50%, 50%”>

</FRAMESET>

Similarly if we replace COLS with ROWS then we get horizontal fragmentation.

The syntax for horizontal fragmentation is,

<FRAMESET ROWS=”50%, 50%”>

</FRAMESET>

**Form:**

The purpose of FORM is to create an HTML form; used to enclose HTML controls, like buttons and text fields.

**Attribute:**

* ACTION: Gives the URL that will handle the form data.
* NAME: Gives the name to the form so you can reference it in code set to an alphanumeric string.
* METHOD: method or protocol is used to sending data to the target action URL. The GET method is the default, it is used to send all form name/value pair information in an URL. Using the POST method, the content of the form are encoded as with the GET method, but are sent in environment variables.

## Controls in HTML:

**<**INPUT TYPE =BUTTON>:

Creates an html button in a form.

ATTRIBUTES:

* NAME: gives the element a name. Set to alphanumeric characters.
* SIZE: sets the size.
* VALUE: sets the caption of the element.

**<**INPUT TYPE = PASSWORD>:

Creates a password text field, which makes typed input.

ATTRIBUTES:

* NAME: gives the element a name, set to alphanumeric characters.
* VALUE: sets the default content of the element.

<INPUT TYPE=RADIO>:

**C**reates a radio button in a form.

ATTRIBUTE:

* NAME: Gives the element a name. Set to alphanumeric character.
* VALUE: Sets the default content of the element.

<INPUT TYPE=SUBMIT>:

Creates a submit button that the user can click to send data in the form back to the web server.

ATTRIBUTES:

NAME: Gives the element a name. Set to alphanumeric characters.

VALUE: Gives this button another label besides the default, Submit Query. Set to alphanumeric characters.

**<**INPUT TYPE=TEXT>:

Creates a text field that the user can enter or edit text in.

ATTRIBUTES:

NAME: Gives the element a name. Set to alphanumeric characters.

VALUE: Holds the initial text in the text field. Set to alphanumeric characters.

**Java Script:**

Java script originally supported by Netscape navigator is the most popular web scripting language today. Java script lets you embedded programs right in your web pages and run these programs using the web browser. You place these programs in a <SCRIPT> element, usually within the <HEAD> element. If you want the script to write directly to the web page, place it in the <BODY> element.

**Java script Methods:**

**Writeln:**

Document.writeln () is a method, which is used to write some text to the current web page.

**onClick:**

Occurs when an element is clicked.

**onLoad:**

Occurs when the page loads.

**onMouseDown:**

Occurs when a mouse button goes down.

**onMouseMove:**

Occurs when the mouse moves.

**OnUnload:**

Occurs when a page is unloaded.

**MySQL:**

MySQL is an open source relational database management system (RDBMS).This is the most popular database system used with PHP. MySQL is distributed and supported by Oracle Corporation.

MySQL runs on almost all platforms including Linux, Unix and Windows. Although it can be used in a wide range of applications, MySQL is often associated with web applications and online publishing.

MySQL is an essential constituent of an open source enterprise stack called LAMP. LAMP is a web development platform that uses Linux as an operating system, in the form of Apache web server, MySQL relational database management system and PHP object-oriented scripting language.

**Advantages of MySQL:**

**Data Security:** MySQL is globally renowned for being the most secure and reliable database management system used in popular web applications including WordPress, Drupal, Joomla, Facebook and Twitter.

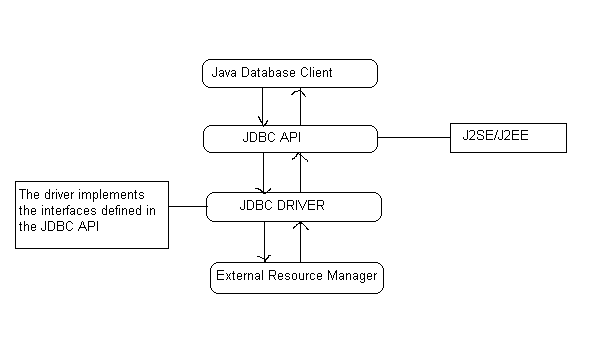
**High Performance**: MySQL features a distinct storage-engine framework that facilitates system administrators to configure the MySQL database server for a flawless performance.

**Round-the-Clock Up-time:** MySQL comes with the assurance of 24×7 up-time and offers a wide range of high-availability solutions, including specialized cluster servers and master/slave replication configurations.

**The Flexibility of Open Source:** All the fears and worries that arise in an open-source solution can be brought to an end with MySQL’s round-the-clock support and enterprise indemnification. The secure processing and trusted software of MySQL combine to provide effective transactions for large-volume projects. It makes maintenance, debugging and upgrades fast and easy while enhancing the end-user experience.

**JDBC Drivers:**

The JDBC API only defines interfaces for objects used for performing various database-related tasks like opening and closing connections, executing SQL commands, and retrieving the results. We all write our programs to interfaces and not implementations. Either the resource manager vendor or a third party provides the implementation classes for the standard JDBC interfaces. These software implementations are called JDBC drivers.JDBC drivers transform the standard JDBC calls to the external resource manager-specific API calls. The diagram below depicts how a database client written in java accesses an external resource manager using the JDBC API

**driver:**

Depending on the mechanism of implementation, JDBC drivers are broadly classified into four types.

**TYPE1:**

Type1 JDBC drivers implement the JDBC API on top of a lower level API like ODBC. These drivers are not generally portable because of the independency on native libraries. These drivers translate the JDBC calls to ODBC calls and ODBC sends the request to external data source using native library calls. The JDBC-ODBC driver that comes with the software distribution for J2SE is an example of a type1 driver.

**TYPE2:**

Type2 drivers are written in mixture of java and native code. Type2 drivers use vendors specific native APIs for accessing the data source. These drivers transform the JDBC calls to vendor specific calls using the vendor’s native library.

These drivers are also not portable like type1 drivers because of the dependency on native code.

**TYPE3:**

Type3 drivers use an intermediate middleware server for accessing the external data sources. The calls to the middleware server are database independent. However, the middleware server makes vendor specific native calls for accessing the data source. In this case, the driver is purely written in java.

**TYPE4:**

Type4 drivers are written in pure java and implement the JDBC interfaces and translate the JDBC specific calls to vendor specific access calls. They implement the data transfer and network protocol for the target resource manager. Most of the leading database vendors provide type4 drivers for accessing their database servers.

**Driver manager and Driver:**

The java.sql package defines an interface called Java.sql.Driver that makes to be implemented by all the JDBC drivers and a class called java.sql.DriverManager that acts as the interface to the database clients for performing tasks like connecting to external resource managers, and setting log streams. When a JDBC client requests the DriverManager to make a connection to an external resource manager, it delegates the task to an approate driver class implemented by the JDBC driver provided either by the resource manager vendor or a third party.

**JAVA.SQL.DRIVERMANAGER:**

The primary task of the class driver manager is to manage the various JDBC drivers register. It also provides methods for:

* Getting connections to the databases.
* Managing JDBC logs.
* Setting login timeout.

**Managing drivers:**

JDBC clients specify the JDBC URL when they request a connection. The driver manager can find a driver that matches the request URL from the list of register drivers and delegate the connection request to that driver if it finds a match JDBC URLs normally take the following format:

**<protocol>:<sub-protocol>:<resource>**

The protocol is always jdbc and the sub-protocol and resource depend on the type of resource manager. The URL for postgreSQL is in the format:

**Jdbc: postgres ://< host> :< port>/<database>**

Here host is the host address on which post master is running and database is the name of the database to which the client wishes to connect.

**Managing controls:**

DriverManager class is responsible for managing connections to the databases:

public static Connection getConnection (String url,Properties info) throws SQLException

This method gets a connection to the database by the specified JDBC URL using the specified username and password. This method throws an instance of SQLException if a database access error occurs.

**Connetions:**

The interface java.sql.Connection defines the methods required for a persistent connection to the database. The JDBC driver vendor implements this interface. A database ‘vendor-neutral’ client never uses the implementation class and will always use only the interface. This interface defines methods for the following tasks:

* Statements, prepared statements, and callable statements are the different types of statements for issuing sql statements to the database by the JDBC clients.
* For getting and setting auto-commit mode.
* Getting meta information about the database.
* Committing and rolling back transactions.

**Creating connections:**

The interface java.sql.Connection defines a set of methods for creating database statements. Database statements are used for sending SQL statements to the database:

Public Statement createStatement () throws SQLException

This method is used for creating instances of the interface java.sql.Statement. This interface can be used for sending SQL statements to the database. The interface java.sql.Statement is normally used for sending SQL statements that don’t take any arguments. This method throws an instance of SQLException if a database access error occurs:

Public Statement createStatement (int resType, int resConcurrency) throws SQLException.

**JDBC resultset:**

A JDBC resultset represents a two dimentional array of data produced as a result of executing SQL SELECT statements against databases using JDBC statements. JDBC resultsets are represented by the interface java.sql.ResultSet. The JDBC vendor provider provides the implementation class for this interface.

**Scrolling resultset:**

public boolean next() throws SQLException

public boolean previous() throws SQLException

public boolean first() throws SQLException

public boolean last() throws SQLException

**Statement:**

The interface java.sql.stament is normally used for sending SQL statements that do not have IN or OUT parameters. The JDBC driver vendor provides the implementation class for this interface. The common methods required by the different JDBC statements are defined in this interface. The methods defined by java.sql. Statement can be broadly categorized as follows:

* Executing SQL statements
* Querying results and resultsets
* Handling SQL batches
* Other miscellaneous methods

The interface java.sql.statements defines

methods for executing different SQL statements like SELECT, UPDATE, INSERT, DELETE, and CREATE.

Public Resultset execute Query (string sql) throws SQLException

The following figure shows how the DriverManager, Driver, Connection, Statement, ResultSet classes are connected.

DriverManager

Driver

Driver

Layer

Application

Layer

Connection

Prepared Statement

Statement

Callable Statement

Result Set

Result Set

Result Set

**JAVA SERVER PAGES (JSP):**

**Introduction:**

Java Server 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 <% and end with %>.

**The need of JSP:**

Servlets are indeed useful, and JSP by no means makes them obsolete. However,

* It is hard to write and maintain the HTML.
* You cannot use standard HTML tools.
* The HTML is inaccessible to non-Java developers.

**Benefits of JSP:**

JSP provides the following benefits over servlets alone:

* It is easier to write and maintain the HTML: In this no extra backslashes, no double quotes, and no lurking Java syntax.
* You can use standard Web-site development tools:

We use Macromedia Dreamweaver for most of the JSP pages. Even HTML tools that know nothing about JSP can used because they simply ignore the JSP tags.

* You can divide up your development team:

The Java programmers can work on the dynamic code. The Web developers can concatenate on the representation layer. On large projects, this division is very important. Depending on the size of your team and the complexity of your project, you can enforce a weaker or stronger separation between the static HTML and the dynamic content.

**Creating template text:**

A large percentage of our JSP document consists of static text known as template text. In almost all respects, this HTML looks just likes normal HTML follows all the same syntax rules, and simply “passed through” to that client by the servlet created to handle the page. Not only does the HTML look normal, it can be created by whatever tools you already are using for building Web pages.

There are two minor exceptions to the “template text passed through” rule. First, if you want to have <% 0r %> in the out port, you need to put <\% or %\> in the template text. Second, if you want a common to appear in the JSP page but not in the resultant document,

<%-- JSP Comment -- %>

HTML comments of the form:

<!—HTML Comment -->

are passed through to the client normally.

**Types of JSP scrolling elements:**

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

1. **Expressions** of the form <%=Java Expression %>, which are evaluated and inserted into the servlet’s output.
2. **Sciptlets** of the form <%Java code %>, which are inserted into the servlet’s\_jspService method (called by service).
3. **Declarations** of the form<%! Field/Method Declaration %>, which are inserted into the body of the servlet class, outside any existing methods.

**Using JSP Expressions:**

A JSP element 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 the information about the request For example, the following shows the date/time that the page was requested.

Current time: <%=new java.util.Date () %>

**Predefined variables:**

To simplify expressions we can use a number of predefined variables (or “implicit objects”). The specialty of these variables is that, the system simple tells what names it will use for the local variables in \_jspService.The most important ones of these are:

* **request**, the HttpServletRequest.
* **response**, the HttpServletResponse.
* **session,** the HttpSession associated with the request
* **out,** the writer used to send output to clients.
* **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.

Here is an example:

Your hostname: <%= **request.**getRemoteHost () %>

**Comparing servlets to JSP pages:**

JSP works best when the structure of the HTML page is fixed but the values at various places need to be computed dynamically. If the structure of the page is dynamic, JSP is less beneficial. Some times servlets are better in such a case. If the page consists of binary data or has little static content, servlets are clearly superior. Sometimes the answer is neither servlets nor JSP alone, but rather a combination of both.

**Writing scriptlets:**

If you want to do something more complex than output the value of a simple expression .JSP scriptlets let you insert arbitrary code into the servlet’s \_jspService method. Scriptlets have the following form:

<% Java code %>

Scriptlets have access to the same automatically defined variables as do expressions (request, response, session, out , etc ) .So for example you want to explicitly send output of the resultant page , you could use the out variable , as in the following example:

<%

String queryData = request.getQueryString ();

out.println (“Attached GET data: “+ queryData);

%>

**Scriptlet Examples:**

As an example of code that is too complex for a JSP expression alone, a JSP page that uses the bgColor request parameter to set the background color of the page .Simply using

<BODY BGCOLOR=”<%= request.getParameter (“bgcolor”) %> “>

would violate the cardinal rule of reading form data.

**Using declarations:**

A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class .A declaration has the following form:

<%! Field or Method Definition %>

Since declarations do not generate output, they are normally used in conjunction with JSP expressions or scriptlets. In principle, JSP declarations can contain field (instance variable) definitions, method definitions, inner class definitions, or even static initializer blocks: anything that is legal to put inside a class definition but outside any existing methods. In practice declarations almost always contain field or method definitions.

We should not use JSP declarations to override the standard servlet life cycle methods. The servlet into which the JSP page gets translated already makes use of these methods. There is no need for declarations to gain access to service, doget, or dopost, since calls to service are automatically dispatched to \_jspService , which is where code resulting from expressions and scriptlets is put. However for initialization and cleanup, we can use jspInit and jspDestroy- the standard init and destroy methods are guaranteed to call these methods in the servlets that come from JSP.

**Jakarta Tomcat:**

Tomcat is the Servlet/JSP container. Tomcat implements the Servlet 2.4 and JavaServer Pages 2.0 specification. It also includes many additional features that make it a useful platform for developing and deploying web applications and web services.

**Terminology:**

Context – a Context is a web application.

$CATALINA\_HOME – This represents the root of Tomcat installation.

**Directions and files:**

**/bin** – Startup, shutdown, and other scripts. The \*.sh files (for Unix systems) are functional duplicates of the \*.bat files (for Windows systems). Since the Win32 command-line lacks certain functionality, there are some additional files in here.

**/conf –** Configuration files and related DTDs. The most important file in here is server.xml. It is the main configuration file for the container.

**/logs –** Log files are here by default.

**/webapps –** This is where webapps go\

**Installation:**

Tomcat will operate under any Java Development Kit (JDK) environment that provides a JDK 1.2 (also known as Java2 Standard Edition, or J2SE) or later platform. JDK is needed so that servlets, other classes, and JSP pages can be compiled.

**Deployment directions for default web applications:**

**HTML and JSP Files**

* Main Location

$CATALINA\_HOME/webapps/ROOT

* Corresponding URLs.

http://host/SomeFile.html

http://host/SomeFile.jsp

* More Specific Location (Arbitrary Subdirectory).

$CATALINA\_HOME/webapps/ROOT/SomeDirectory

* Corresponding URLs

http://host/SomeDirectory/SomeFile.html

http://host/SomeDirectory/SomeFile.jsp

**Individual Servlet and Utility Class Files**

* Main Location (Classes without Packages).

$CATALINA\_HOME/webapps/ROOT/WEB-INF/classes

* Corresponding URL (Servlets).

http://host/servlet/ServletName

* More Specific Location (Classes in Packages).

$CATALINA\_HOME/webapps/ROOT/WEB-INF/classes/packageName

* Corresponding URL (Servlets in Packages).

http://host/servlet/packageName.ServletName

**Servlet and Utility Class Files Bundled in JAR Files**

* Location

$CATALINA\_HOME/webapps/ROOT/WEB-INF/lib

* Corresponding URLs (Servlets)

http://host/servlet/ServletName

http://host/servlet/packageName.ServletName

**XAMPP:**

XAMPP:

XAMPP stands for Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P).

It is simply a web server if we want to make a website or designing and make a working website then XAMPP is useful .it gives an environment of hiw server works.

1. It contains apache, mysql, filezilla servers by which we can use them and helps us in login and logout sessions, cookies we gives a good help in websites

2. Also, it has wordpress feature by which it contains many themes of websites which are popular and we can use them to make a website without using so much php coding ,HTML, CSS etc.

3.How to use it: 1. if we are working on mysql then we just on the server of mysql and go to php admin page.

4.To work on php based web pages we just on the server and then, code on a notepad by using php pages.