

UNIT 4

The Server Tier

(8 Hours) — Web Technology (CACS205)

4.1 Web Server Concept

A **web server** is software (running on a computer) that accepts HTTP requests from clients — normally browsers — and returns HTTP responses such as HTML pages, images, or the output of programs. Popular web servers are **Apache HTTP Server**, **Nginx**, **Microsoft IIS** and **Apache Tomcat** (a servlet/JSP container).



Dashed arrows: results flow back → dynamic HTML response returned to the browser

Figure 4.1 — Request/response cycle in the server tier

4.1.1 The HTTP Request–Response Cycle

- The browser opens a TCP connection to the server (default port 80 for HTTP, 443 for HTTPS).
- It sends a **request**: request line (method, URL, version), headers, and optional body.
- The web server maps the URL to a file or program. A *static* resource is read from disk; a *dynamic* resource is passed to a server-side program.
- The program runs, possibly querying a database, and produces output (usually HTML).
- The server sends a **response**: status line, headers (Content-Type, Set-Cookie ...) and the body.

HTTP Method	Purpose
GET	Retrieve a resource; parameters in URL; safe and idempotent.
POST	Send data in the body to be processed (forms, uploads).

HTTP Method	Purpose
PUT / DELETE	Create-replace / remove a resource.
HEAD	Like GET but returns headers only.

Status code	Meaning
200 OK	Request succeeded.
301 / 302	Resource moved permanently / temporarily (redirect).
400 Bad Request	Malformed request.
401 / 403	Unauthorised (login needed) / Forbidden.
404 Not Found	Resource does not exist.
500 Internal Server Error	The server-side program failed.

Static Content	Dynamic Content
Same file returned to every user	Generated at request time; differs per user/request
Plain HTML, CSS, images	Servlets, JSP, PHP, ASP.NET
Fast, easy to cache	Needs processing, can use database and session data

4.2 Creating Dynamic Content

Dynamic content is produced by server-side programs. Common technologies are CGI, PHP, ASP.NET, and Java's **Servlets** and **JavaServer Pages (JSP)**. A **servlet** is a Java class running inside a servlet container that handles requests; a **JSP** is an HTML page with embedded Java that the container compiles automatically into a servlet.

4.2.1 Servlet Example

HelloServlet.java

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

public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        String name = req.getParameter("name"); // form data
        PrintWriter out = res.getWriter();
        out.println("<html><body>");
        out.println("<h1>Hello, " + name + "!</h1>");
        out.println("</body></html>");
    }
}
```

```
}
}
```

4.2.2 JSP Elements

Element	Syntax	Purpose
Directive	<code><%@ page import="java.util.*" %></code>	Instructions to the container (page, include, taglib).
Declaration	<code><%! int count = 0; %></code>	Declares variables/methods at class level.
Scriptlet	<code><% code %></code>	Java statements run for each request.
Expression	<code><%= expr %></code>	Evaluates and prints a value into the page.
Comment	<code><%-- hidden --%></code>	Not sent to the client.
Action	<code><jsp:include page="a.jsp"/></code>	Standard XML-style actions (include, forward, useBean).

Implicit object	Type / purpose
request	HttpServletRequest — parameters, headers, cookies from the client.
response	HttpServletResponse — status, headers, output.
session	HttpSession — data kept for one user across requests.
application	ServletContext — data shared by all users of the web app.
out	JspWriter — writes to the response body.
exception	Throwable — available on error pages.

4.3 Control Flow in Dynamic Content Generation

Ordinary Java control statements — if/else, switch, for, while — can be mixed with HTML in a JSP so that parts of the page are generated conditionally or repeated. Only the HTML inside the branch that is executed is sent to the browser.

Conditional and repeated output in JSP

```
<%@ page language="java" %>
<html><body>
<%
    int hour = java.time.LocalTime.now().getHour();
    if (hour < 12) {
%>
    <h2>Good morning!</h2>
<% } else { %>
    <h2>Good afternoon / evening!</h2>
```

```
<% } %>

<table border="1">
<% for (int i = 1; i <= 5; i++) { %>
    <tr><td><%= i %></td><td><%= i * i %></td></tr>
<% } %>
</table>
</body></html>
```

Note: Too much Java inside JSP pages becomes unreadable. This is why the JSP Standard Tag Library (JSTL) with tags such as `c:if` and `c:forEach` is preferred (see Section 4.7).

4.4 Sessions and State

HTTP is a **stateless** protocol: each request is independent and the server does not automatically remember earlier requests. Applications like shopping carts and logins need **state**. A **session** is a series of related requests from the same user, tracked by the server.

Technique	How it works	Notes
Cookies	Server sends Set-Cookie; the browser returns it with each request.	Stored on client; size-limited; user can disable.
URL rewriting	Session id appended to every URL (...;jsessionid=AB12).	Works without cookies; id visible in URL.
Hidden form fields	State placed in hidden inputs and resubmitted.	Only for form-to-form flows.
Server-side session (HttpSession)	Data kept on the server, keyed by a session id cookie.	Most common; secure; uses server memory.
Database / application scope	State persisted in database or shared by all users.	Survives restarts / shared data.

Using HttpSession

```
<%
    // store data in the session
    session.setAttribute("user", "ram");
    session.setMaxInactiveInterval(15 * 60); // 15 minutes

    // read it back on any later request
    String u = (String) session.getAttribute("user");
    if (u == null) { response.sendRedirect("login.jsp"); }

    // logout: destroy the session
    session.invalidate();
%>
```

Scope	Lifetime	Object
page	Current page only	pageContext

Scope	Lifetime	Object
request	One request (and forwards)	request
session	Until the user's session ends or times out	session
application	Until the web application stops	application

4.5 Error Handling

Server-side programs can fail (bad input, missing database, programming mistakes). Good error handling prevents the user from seeing raw stack traces and helps developers diagnose the fault. The main mechanisms are:

- **try / catch / finally** blocks around code that may throw exceptions.
- **JSP error pages:** a page declares `<%@ page errorPage="error.jsp" %>` and the error page declares `<%@ page isErrorPage="true" %>`.
- **web.xml error mapping** by HTTP status code (404, 500) or by exception type.
- **Logging** of errors to files for later analysis; **validation** of user input to avoid errors in the first place.

JSP error page

```

<!-- page that may fail -->
<%@ page errorPage="error.jsp" %>
<% int n = Integer.parseInt(request.getParameter("age")); %>

<!-- error.jsp -->
<%@ page isErrorPage="true" %>
<h2>Sorry, something went wrong.</h2>
<p>Reason: <%= exception.getMessage() %></p>

```

Declaring error pages in web.xml

```

<!-- web.xml -->
<error-page>
  <error-code>404</error-code>
  <location>/notfound.html</location>
</error-page>
<error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/error.jsp</location>
</error-page>

```

4.6 Architecting Web Applications

As applications grow, mixing HTML, business logic and database code in one JSP becomes unmaintainable. Web applications are therefore structured with the **Model–View–Controller**

(MVC) pattern.

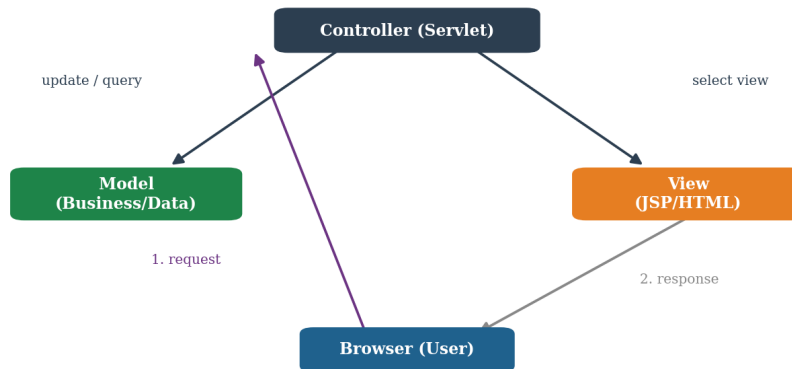


Figure 4.2 — MVC architecture for a web application

MVC component	Role	Java web technology
Model	Business data and rules; talks to the database.	JavaBeans, DAO classes, EJB
View	Presents data to the user; contains no business logic.	JSP with JSTL / EL, HTML
Controller	Receives requests, calls the model, chooses the next view.	Servlet (or a framework such as Struts, Spring MVC)

Flow: the browser request reaches the *controller servlet*; it validates input and calls the *model*; the result is stored in the request/session; the controller *forwards* to a *JSP view* that formats the output. Benefits: separation of concerns, easier testing, parallel work by designers and programmers, and reusable components. Other design guidelines: use a layered package structure (web, service, dao), keep configuration in web.xml, centralise error handling and security, and use connection pools.

Typical web-application directory layout

```
MyApp/
├── WEB-INF/
│   ├── web.xml (deployment descriptor)
│   ├── classes/ (servlets, beans, DAO)
│   ├── lib/ (jar files)
│   └── tlds/ (tag library descriptors)
├── index.jsp (views)
└── css/ images/ js/
```

4.7 Using Tag Libraries

A **tag library** packages reusable actions as custom XML-style tags so JSP pages can avoid Java scriptlets. The standard one is **JSTL (JSP Standard Tag Library)**. A page imports a library

with the `taglib` directive, which assigns a *prefix* and identifies the library by its URI.

JSTL library	Prefix	Examples of tags
Core	c	c:out, c:set, c:if, c:choose/when/otherwise, c:forEach, c:redirect, c:import
Formatting	fmt	fmt:formatNumber, fmt:formatDate, fmt:message
SQL	sql	sql:setDataSource, sql:query, sql:update
XML	x	x:parse, x:out, x:forEach
Functions	fn	fn:length, fn:contains, fn:toUpperCase

Using JSTL core tags with Expression Language (EL)

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html><body>
  <c:set var="marks" value="72"/>
  <c:if test="${marks} >= 40">
    <p>Pass – marks: <c:out value="${marks}"/></p>
  </c:if>

  <c:choose>
    <c:when test="${marks} >= 80">Distinction</c:when>
    <c:when test="${marks} >= 60">First Division</c:when>
    <c:otherwise>Second Division</c:otherwise>
  </c:choose>

  <ul>
    <c:forEach var="s" items="${students}">
      <li>${s.name}</li>
    </c:forEach>
  </ul>
</body></html>
```

4.8 Writing Tag Libraries

Developers can create their own **custom tags**. Three pieces are needed: a **tag handler** class (Java), a **Tag Library Descriptor (TLD)** file that describes the tags, and the **taglib directive** in the JSP.

Step 1 — tag handler

```
// 1. Tag handler: HelloTag.java
package mytags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.IOException;

public class HelloTag extends SimpleTagSupport {
    private String name; // attribute
    public void setName(String name) { this.name = name; }
```

```

    public void doTag() throws JspException, IOException {
        getJspContext().getOut().print("Hello, " + name + "!");
    }
}

```

Steps 2 and 3 — descriptor and use

```

<!-- 2. TLD: WEB-INF/tlds/mytags.tld -->
<taglib>
  <tlib-version>1.0</tlib-version>
  <short-name>my</short-name>
  <uri>http://example.com/mytags</uri>
  <tag>
    <name>hello</name>
    <tag-class>mytags.HelloTag</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <name>name</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>

<!-- 3. Use it in a JSP -->
<%@ taglib uri="http://example.com/mytags" prefix="my" %>
<my:hello name="Sita"/>

```

Handler type	Use
SimpleTagSupport	Simple tags: override doTag() (recommended).
TagSupport	Classic tags without body iteration: doStartTag(), doEndTag().
BodyTagSupport	Classic tags that read or change their body: doAfterBody().

Example 4.1 — Session counter

Write a JSP fragment that counts how many times a user has visited the page in the current session.

Solution

Store an Integer in the session: read `session.getAttribute("visits")`; if it is null set it to 1, otherwise add 1; save it back with `setAttribute` and print it with `<%= visits %>`. Because the value lives in the session (not the request), it survives across page loads for the same user.

Key Takeaways — Unit 4

- ✓ A web server maps HTTP requests to static files or to server-side programs and returns HTTP responses.
- ✓ Dynamic content is created by servlets and JSP; JSP scriptlets, expressions and directives mix Java with HTML.
- ✓ Control flow statements decide which HTML fragments are generated or repeated.
- ✓ HTTP is stateless; cookies, URL rewriting, hidden fields and HttpSession maintain state.
- ✓ Use try/catch, `errorPage/isErrorPage` and `web.xml` error-page entries for robust error handling.
- ✓ MVC separates model, view and controller; JSTL tag libraries remove scriptlets, and custom tags need a handler class, a TLD and a taglib directive.

— *End of Unit 4* —

bcalink.com