Web Technology / Internet Programming – Unit Notes

Server-Side Architecture • Dynamic Web Applications • Java Servlets and JSP

1. Server-Side Architecture

1.1 Introduction

A web application works on the request–response model. A client (normally a web browser) sends a request over the Internet using the HTTP protocol, and a program running on a remote machine – the server – processes the request and sends back a response. Everything that happens on the server before the response is produced is called server-side processing, and the design that organises these components is called the server-side architecture.

In server-side processing the web page is generated on the server. The browser only receives the final HTML, CSS and JavaScript, so the client does not need to know how the page was produced or where the data came from.

1.2 Client–Server Model

- **Client:** requests services; it presents the user interface (browser, mobile app).
- **Server:** receives requests, runs the business logic, accesses data and returns the result.
- **Protocol:** HTTP/HTTPS over TCP/IP is used for communication. HTTP is stateless – the server does not remember earlier requests by default.

1.3 Three-Tier Architecture

Modern web applications are divided into three logical tiers so that each part can be developed, scaled and maintained independently.

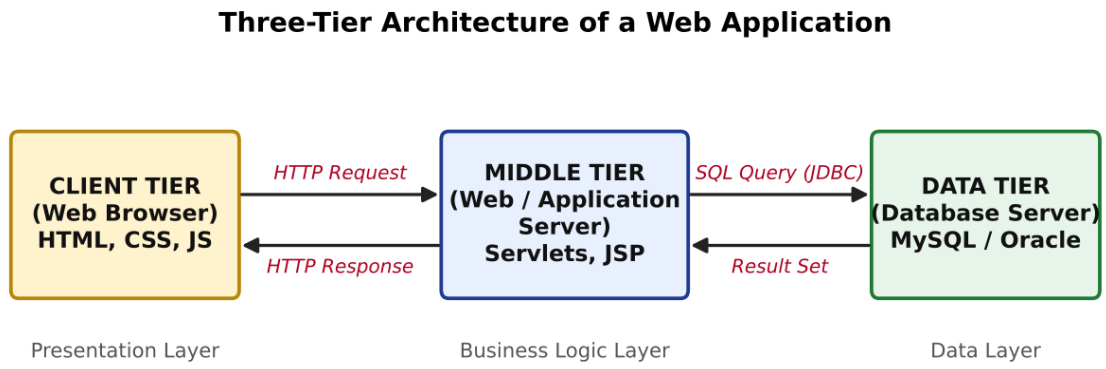


Fig. 1 Three-tier architecture of a web application

Tier	Role	Technologies
Presentation (Client) tier	Displays the user interface and accepts user input	HTML, CSS, JavaScript, browser

Tier	Role	Technologies
Middle (Application) tier	Processes requests, applies business rules, controls flow	Servlets, JSP, Tomcat, Java EE
Data tier	Stores and retrieves persistent data	MySQL, Oracle, PostgreSQL via JDBC

1.3.1 Advantages of the three-tier design

- Separation of concerns – UI, logic and data are independent.
- Scalability – each tier can be scaled or load-balanced separately.
- Security – the browser never talks directly to the database.
- Reusability and easy maintenance of the business logic.

1.4 Web Server vs Application Server

Web Server	Application Server
Handles HTTP requests and serves static content (HTML, images, CSS)	Runs business logic and generates dynamic content
Cannot run Java components by itself	Contains containers for Servlets, JSP, EJB
Examples: Apache HTTP Server, Nginx, IIS	Examples: Apache Tomcat (web container), JBoss/WildFly, GlassFish, WebLogic
Lightweight, very fast for static pages	Provides transactions, security, connection pooling

1.5 Server-Side Technologies

Common technologies used to build the middle tier are CGI scripts, PHP, ASP.NET, Node.js, and in the Java world Servlets, JSP, JSF and Spring. This unit concentrates on Java Servlets and JSP, which together form the foundation of Java EE (now Jakarta EE) web development.

2. Dynamic Web Applications

2.1 Static vs Dynamic Web Pages

Static Web Page	Dynamic Web Page
Content is fixed and stored in an HTML file	Content is generated at request time
Same page is shown to every user	Page depends on user, time, input or database data
No server-side programming needed	Needs server-side programs (Servlet, JSP, PHP, etc.)
Changing content means editing the file	Content changes automatically with the data

Static Web Page	Dynamic Web Page
Example: About-us page	Example: Online banking, shopping cart, result portal

2.2 What is a Dynamic Web Application?

A dynamic web application is a web-based program whose pages are created on the fly using server-side logic. It accepts input from the user (forms, links, URL parameters), processes the input, interacts with databases or other services and builds a customised response.

Key features

- User interaction through forms and events.
- Database connectivity for storing and retrieving data.
- Session management to remember users across requests.
- Authentication, authorisation and personalised content.

2.3 Request Processing in a Dynamic Web Application

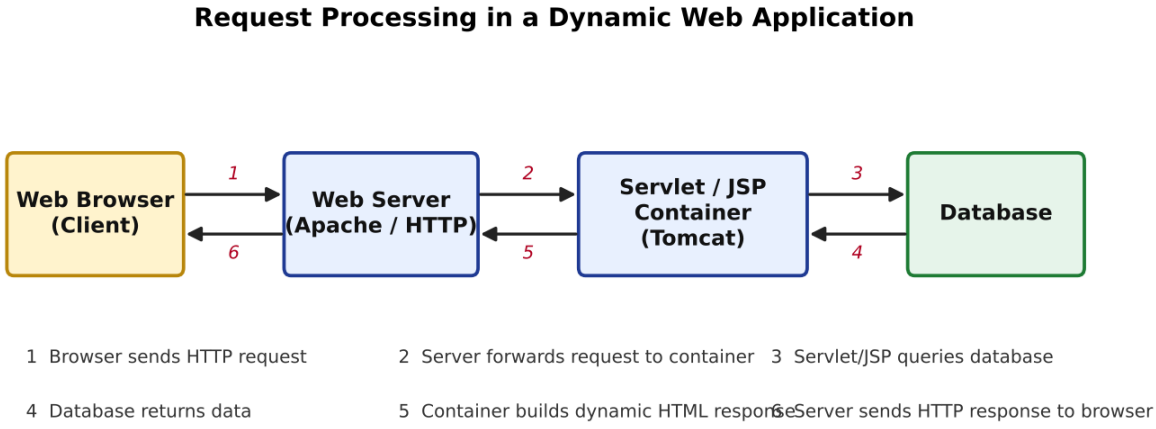


Fig. 2 Request processing in a dynamic web application

1. The user enters a URL or submits a form; the browser sends an HTTP request.
2. The web server receives the request. Static resources are returned directly; dynamic requests are passed to the servlet/JSP container.
3. The container maps the URL to a Servlet or JSP and runs it.
4. The program connects to the database (JDBC), executes queries and receives the data.
5. The program builds an HTML response using the data.
6. The web server sends the HTTP response back and the browser displays the page.

2.4 HTTP Methods: GET and POST

Feature	GET	POST
Data location	Appended to the URL as query string	Sent in the request body
Visibility	Visible in URL and browser history	Not visible in URL
Size limit	Limited by URL length	No practical limit
Use	Retrieving data, searching, bookmarking	Form submission, login, file upload, data changes
Idempotent	Yes – repeating gives the same result	No – may change server state
Servlet method	doGet()	doPost()

2.5 HTTP Request and Response Structure

- **Request:** request line (method, URL, HTTP version), request headers (Host, User-Agent, Accept, Cookie), blank line, optional body.
- **Response:** status line (e.g. HTTP/1.1 200 OK), response headers (Content-Type, Content-Length, Set-Cookie), blank line, body (HTML).

Status code	Meaning
200	OK – request succeeded
302	Found – redirect to another URL
404	Not Found – resource does not exist
500	Internal Server Error – the server program failed

3. Java Servlets

3.1 Definition

A Servlet is a Java class that runs inside a web server (servlet container) and extends the capabilities of the server by processing client requests and generating dynamic responses. Servlets are platform independent, secure and run inside the JVM.

Note: In Java EE 8 and earlier the packages are javax.servlet.*. From Jakarta EE 9 (Tomcat 10 onwards) they are renamed jakarta.servlet.*. The concepts are identical.

3.2 Servlet Container (Web Container)

The servlet container (for example Apache Tomcat) is the part of the server that manages the life cycle of servlets. It receives the HTTP request, creates the request and response objects, selects the correct servlet, and calls its methods.

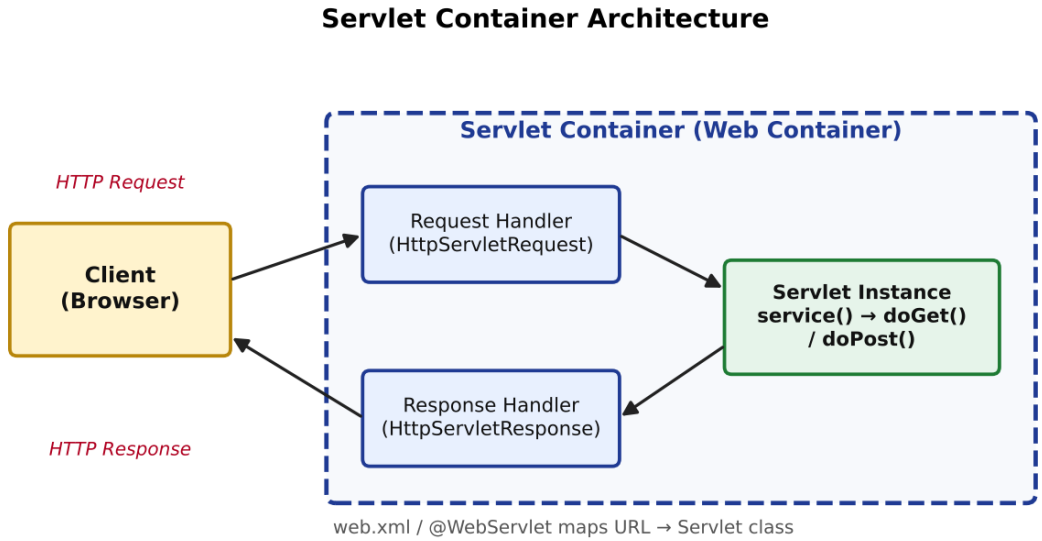


Fig. 3 Servlet container architecture

Services provided by the container

- Communication support – parses HTTP requests and builds responses.
- Life-cycle management – loading, instantiation, initialisation and destruction.
- Multithreading – one thread per request using a single servlet instance.
- Declarative security, JSP support and session management.

3.3 Servlet API and Class Hierarchy

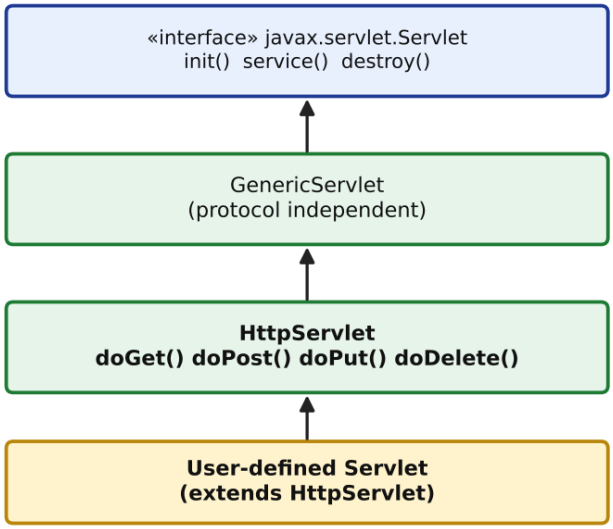


Fig. 4 Servlet class hierarchy

Interface / Class	Description
Servlet	Core interface; declares init(), service(), destroy(), getServletConfig(), getServletInfo()
GenericServlet	Abstract class implementing Servlet; protocol independent

Interface / Class	Description
HttpServlet	Abstract class for HTTP; provides doGet(), doPost(), doPut(), doDelete()
ServletRequest / HttpServletRequest	Carries client request data (parameters, headers, cookies)
ServletResponse / HttpServletResponse	Used to send data, headers and status to the client
ServletConfig	Per-servlet initialisation parameters
ServletContext	Information shared by the whole web application

3.4 Servlet Life Cycle

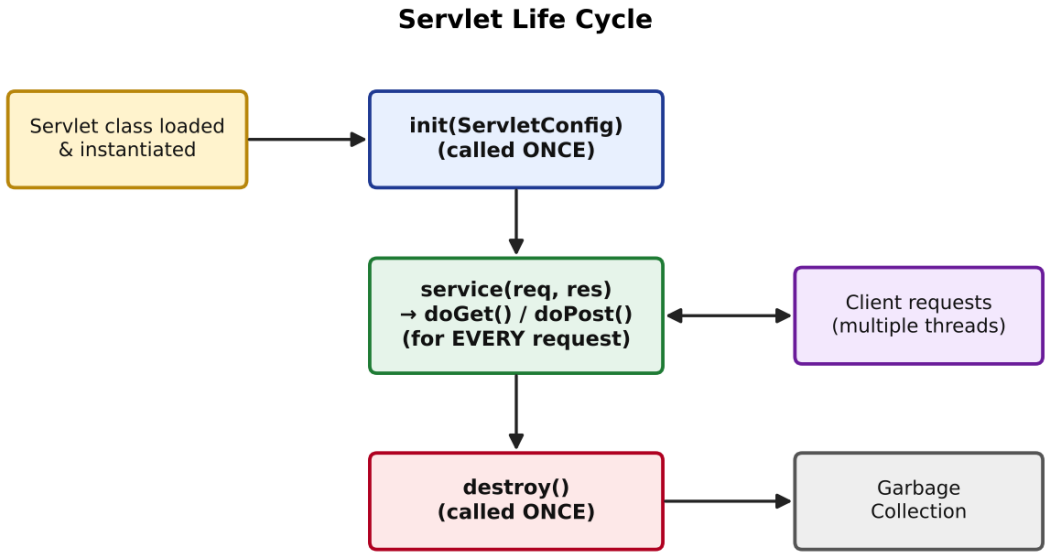


Fig. 5 Servlet life cycle

1. **Loading and instantiation:** the container loads the servlet class and creates one instance, either at start-up or on the first request.
2. **Initialisation – init():** called only once to perform one-time set-up such as reading configuration or opening resources.
3. **Request handling – service():** called for every request. HttpServlet.service() checks the HTTP method and dispatches to doGet(), doPost(), etc. Each request runs in a separate thread.
4. **Destruction – destroy():** called once before the servlet is removed so that resources (connections, files) can be released. The object is then garbage collected.

3.5 A Simple Servlet Program

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
```

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

  protected void doGet(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String name = req.getParameter("user");
    out.println("<html><body>");
    out.println("<h2>Hello, " + name + "</h2>");
    out.println("</body></html>");
  }
}
```

3.6 Deployment Descriptor (web.xml)

The servlet must be mapped to a URL. This is done either with the @WebServlet annotation (Servlet 3.0 and above) or in the deployment descriptor WEB-INF/web.xml:

```
<web-app>
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>HelloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Standard web application structure: the WEB-INF folder (not directly accessible by the client) contains web.xml, a classes folder for compiled .class files and a lib folder for JAR files; HTML and JSP files are kept in the application root.

3.7 Reading Request Data and Sending Response

Method	Purpose
getParameter(String name)	Returns the value of a form field or query parameter
getParameterValues(String name)	Returns all values (e.g. check boxes)
getHeader(String name)	Returns the value of a request header
getCookies()	Returns the cookies sent by the client
getSession()	Returns the current HttpSession (creates one if none exists)
setContentType(String)	Sets the MIME type of the response, e.g. text/html
getWriter()	Returns a PrintWriter to write character data
sendRedirect(String url)	Redirects the browser to another URL
setStatus(int) / sendError(int)	Sets the HTTP status code

3.8 Session Tracking

HTTP is stateless, so a web application must use a technique to identify that a series of requests come from the same user. This is called session tracking (session management).

Technique	Description	Drawback
Cookies	Small text data stored in the browser and returned with each request	User can disable cookies
URL rewriting	Session ID appended to the URL (;jsessionid=...)	Ugly URLs, security risk
Hidden form fields	Data stored in <input type="hidden"> fields	Works only with form submission
HttpSession	Server-side object identified by a JSESSIONID cookie	Uses server memory

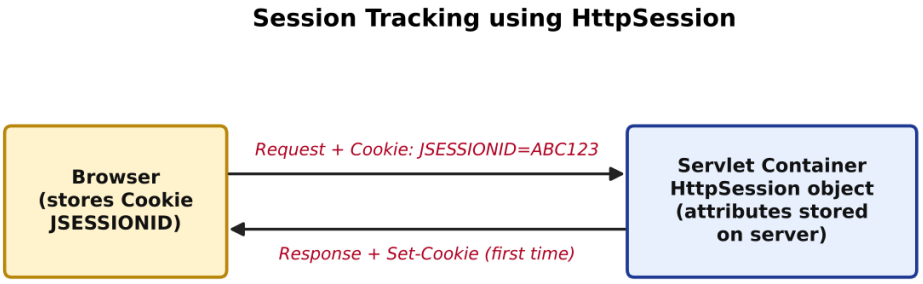


Fig. 6 Session tracking using HttpSession

```
HttpSession session = request.getSession();   // create or get
session.setAttribute("user", "Ravi");          // store
String u = (String) session.getAttribute("user"); // retrieve
session.setMaxInactiveInterval(900);           // timeout in seconds
session.invalidate();                          // logout
```

```
Cookie c = new Cookie("theme", "dark");
c.setMaxAge(60 * 60 * 24);   // one day
response.addCookie(c);
```

3.9 ServletConfig vs ServletContext

ServletConfig	ServletContext
One object per servlet	One object per web application
Holds init parameters of that servlet (<init-param>)	Holds application-wide parameters (<context-param>)
Obtained using getServletConfig()	Obtained using getServletContext()
Not shared between servlets	Shared by all servlets and JSPs

3.10 Servlet Collaboration (RequestDispatcher)

forward()	include()
Control is transferred to another resource	Output of another resource is added to the current response
Original response content is discarded	Original servlet continues after the include
Browser URL does not change	Browser URL does not change

```
RequestDispatcher rd = request.getRequestDispatcher("result.jsp");
rd.forward(request, response);
```

sendRedirect() differs from forward(): it makes the browser send a new request, so the URL changes and request attributes are lost.

3.11 Database Access using JDBC in a Servlet

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection(
      "jdbc:mysql://localhost:3306/college", "root", "password");
PreparedStatement ps = con.prepareStatement(
      "SELECT name FROM student WHERE regno = ?");
ps.setString(1, request.getParameter("regno"));
ResultSet rs = ps.executeQuery();
while (rs.next()) out.println(rs.getString("name"));
con.close();
```

PreparedStatement is preferred over Statement because it prevents SQL injection and is faster for repeated queries.

3.12 Servlet vs CGI

Servlet	CGI
Runs as a thread inside the JVM	A new process is created for every request
Fast and scalable	Slow; heavy on server resources
Platform independent (Java)	Depends on language and platform
Can share data and resources between requests	Cannot easily share resources
Container provides security and sessions	Programmer must implement them

4. JavaServer Pages (JSP)

4.1 Introduction

JavaServer Pages (JSP) is a server-side technology that lets developers embed Java code in HTML pages using special tags. Writing HTML inside println() statements in a servlet is

tedious; JSP solves this by allowing the designer to write normal HTML and place the dynamic parts inside JSP tags. Internally, every JSP is translated into a servlet by the container.

Advantages of JSP

- Easier to write and maintain than servlets for presentation.
- Separates presentation (HTML) from business logic (Java).
- Gets all benefits of servlets – platform independence, performance and security.
- Supports custom tags, JSTL and Expression Language (EL).

4.2 JSP Life Cycle

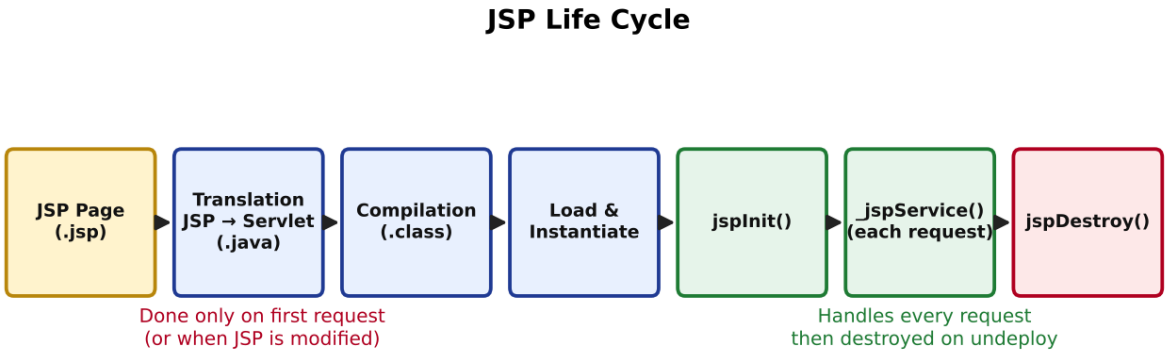


Fig. 7 JSP life cycle

1. **Translation:** the JSP file is converted into a servlet source file (.java).
2. **Compilation:** the generated servlet is compiled into a .class file.
3. **Class loading and instantiation:** the class is loaded and an object is created.
4. **Initialisation – jspInit():** called once.
5. **Request processing – _jspService():** called for every request; it must not be overridden by the programmer.
6. **Destruction – jspDestroy():** called once when the page is removed or the server stops.

The first request to a JSP is slower because of translation and compilation; later requests are served by the already loaded servlet.

4.3 JSP Elements

(a) Directives – <%@ ... %>

Give instructions to the container about the whole page.

Directive	Purpose	Example
page	Page-level settings: language, import, errorPage, session	<%@ page import="java.util.*" %>

Directive	Purpose	Example
include	Includes another file at translation time (static include)	<%@ include file="header.html" %>
taglib	Declares a tag library such as JSTL	<%@ taglib uri="..." prefix="c" %>

(b) Scripting elements

Element	Syntax	Description
Declaration	<%! int count = 0; %>	Declares variables/methods at class level
Scriptlet	<% ... %>	Java statements placed inside _jspService()
Expression	<%= expression %>	Evaluates and prints the value to output
Comment	<%-- comment --%>	JSP comment, not sent to the client

(c) Standard actions – <jsp:... />

Action	Purpose
<jsp:include page="..."/>	Includes another resource at request time (dynamic include)
<jsp:forward page="..."/>	Forwards the request to another resource
<jsp:useBean id="..." class="..." scope="..."/>	Creates or locates a JavaBean
<jsp:setProperty .../>	Sets a property of a bean
<jsp:getProperty .../>	Reads a property of a bean

4.4 JSP Implicit Objects

Implicit objects are created automatically by the container and are available in every JSP without declaration.

Object	Type	Description
request	HttpServletRequest	Client request data
response	HttpServletResponse	Response sent to the client
out	JspWriter	Writes output to the page
session	HttpSession	Session data of the user
application	ServletContext	Application-wide data
config	ServletConfig	Servlet configuration
pageContext	PageContext	Access to all scopes and page attributes
page	Object	Reference to the current JSP servlet (this)
exception	Throwable	Available only in error pages

4.5 Sample JSP Page

```
<%@ page language="java" import="java.util.Date" %>
<html>
<body>
  <h2>Welcome, <%= request.getParameter("user") %></h2>
  <%
     Date d = new Date();
     out.println("Current time: " + d);
  %>
</body>
</html>
```

4.6 Scopes of JSP Objects

Scope	Lifetime
page	Only the current page
request	Until the response is sent (including forwards)
session	Across multiple requests of the same user
application	Entire web application until the server stops

4.7 Expression Language (EL) and JSTL

EL simplifies access to data using the syntax ${expression}, for example ${user.name} or ${param.id}. The JSP Standard Tag Library (JSTL) provides ready-made tags so that scriptlets can be avoided.

```
<%@ taglib uri="jakarta.tags.core" prefix="c" %>
<c:if test="${marks >= 50}">Pass</c:if>
<c:forEach var="s" items="${students}">
   <li>${s.name}</li>
</c:forEach>
```

4.8 Servlet vs JSP

Servlet	JSP
Java code with HTML inside println()	HTML with Java code inside tags
Best for control and business logic	Best for presentation / view
Must be compiled manually	Translated and compiled automatically by the container
Harder to maintain for HTML output	Easy for web designers
Has no implicit objects	Has nine implicit objects

4.9 MVC Architecture using Servlets and JSP

The best practice is to combine both technologies using the Model–View–Controller (Model 2) pattern.

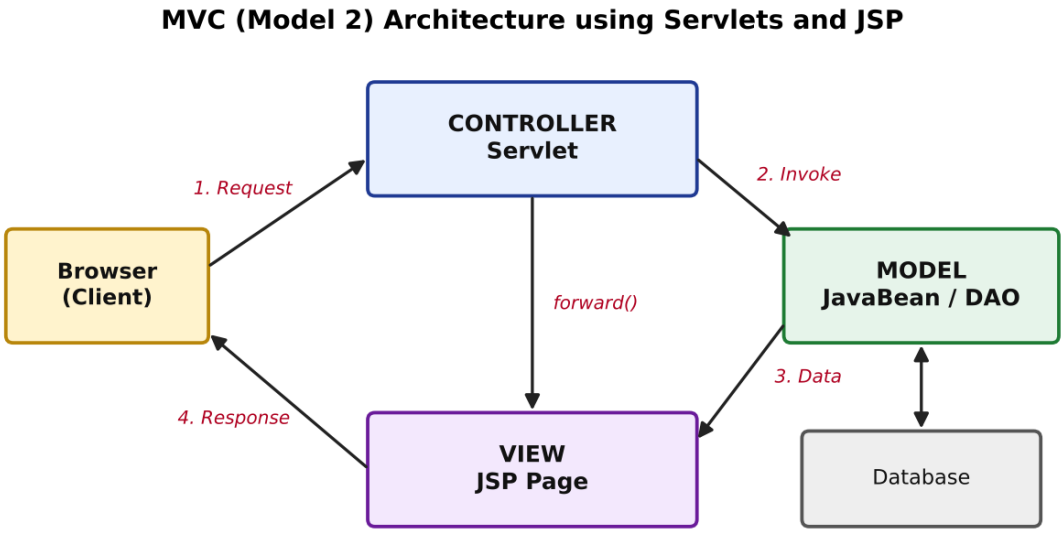


Fig. 8 MVC (Model 2) architecture

- **Model:** JavaBeans and DAO classes that represent data and talk to the database.
- **View:** JSP pages that display the data to the user.
- **Controller:** a servlet that receives the request, calls the model and forwards the result to the proper view.

Flow: the browser sends a request to the controller servlet → the servlet calls the model → the model reads or updates the database → the servlet stores the result in a request attribute and forwards to a JSP → the JSP generates the HTML response.