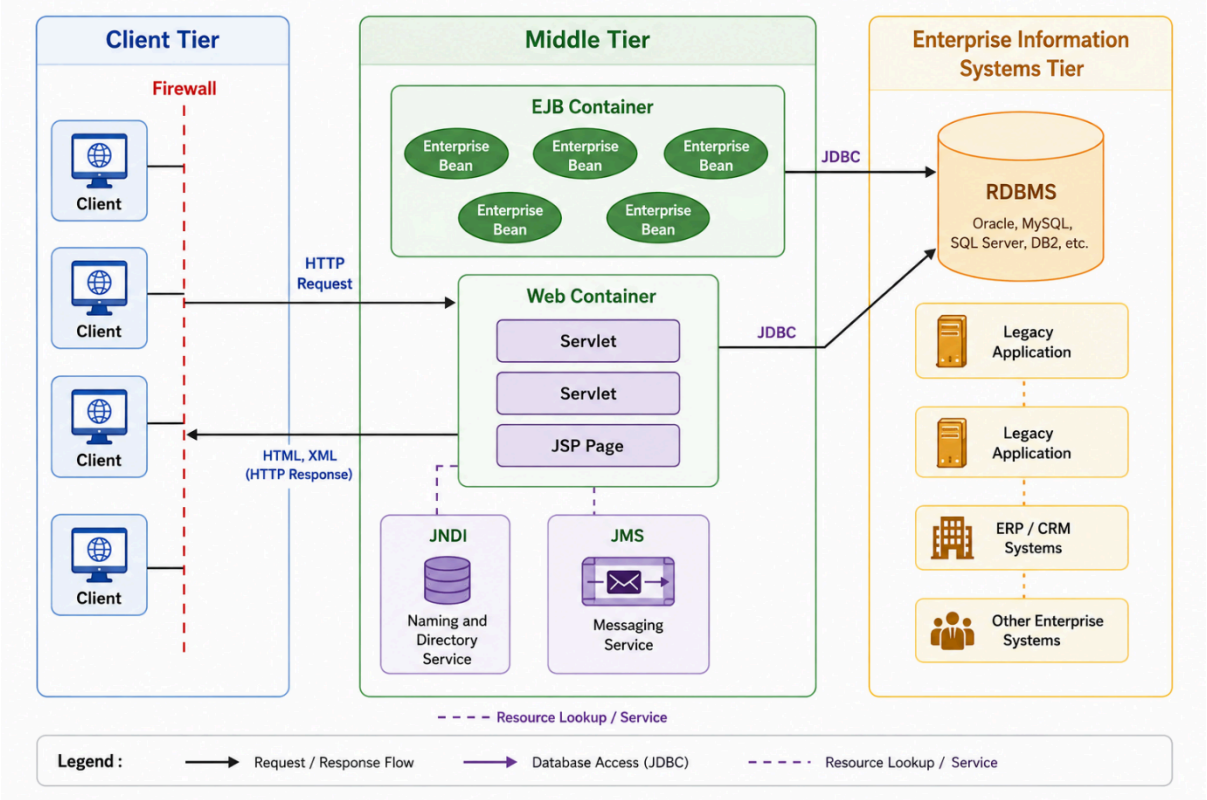
Unit-II: J2EE and Web Development

J2EE Architecture (Enterprise Architecture)



The **J2EE (Java 2 Enterprise Edition) Multi-Tier Architecture** divides an enterprise application into multiple layers (tiers). Each tier performs a specific responsibility, improving **modularity, scalability, security, maintainability, and reusability**. The three major tiers are the **Client Tier**, **Middle Tier**, and **Enterprise Information Systems (EIS) Tier**.

1. Client Tier (Presentation Layer)

The **Client Tier** represents the users who interact with the application. Clients access the application through a web browser, desktop application, or mobile device.

Components

- Web Browser
- Desktop Client
- Mobile Application

Responsibilities

- Sends HTTP requests to the web server.
- Receives HTML/XML/JSON responses.
- Displays the user interface.
- Collects user input.

Communication

- HTTP/HTTPS

2. Middle Tier (Application Server)

The **Middle Tier** contains the application logic and acts as the bridge between the client and the database.

It consists of several important components.

A. Web Container

The **Web Container** manages web components such as **Servlets** and **JSP pages**.

Components

- Servlets
- JSP Pages

Responsibilities

- Receives HTTP requests from clients.
- Processes client requests.
- Calls business logic.
- Generates dynamic web pages.
- Sends responses back to the client.

Technologies

- Servlets
- JSP
- JSTL
- EL

B. EJB Container (Enterprise JavaBeans)

The **EJB Container** manages enterprise beans that contain the application's business logic.

Components

- Enterprise Beans
- Session Beans
- Message-Driven Beans

Responsibilities

- Implements business logic.
- Transaction management.
- Security.
- Thread management.
- Connection pooling.
- Remote method invocation.

Advantages

- Reusable business components.
- Better scalability.
- Automatic transaction management.

C. JNDI (Java Naming and Directory Interface)

JNDI provides a naming service for locating enterprise resources.

Used for locating

- Database connections
- EJB components
- Mail sessions
- JMS resources

Example

```
Context ctx = new InitialContext();
```

DataSource ds =(DataSource) ctx.lookup("jdbc/MyDB");

D. JMS (Java Message Service)

JMS enables asynchronous communication between applications.

Features

- Message Queue
- Publish/Subscribe
- Reliable messaging

Applications

- Banking
- E-commerce
- Order Processing
- Notification Systems

3. Enterprise Information Systems (EIS) Tier

The **Enterprise Information Systems Tier** stores permanent business data and integrates external enterprise systems.

Components

- Oracle Database
- MySQL
- SQL Server
- PostgreSQL
- DB2
- ERP Systems
- CRM Systems
- Legacy Applications

Responsibilities

- Stores application data.
- Retrieves records.
- Executes SQL queries.
- Maintains data consistency.

JDBC

JDBC (Java Database Connectivity) is the API used by Servlets and EJBs to communicate with relational databases.

Responsibilities

- Open database connections.
- Execute SQL statements.
- Retrieve records.
- Update/Delete data.
- Close database connections.

Advantages of J2EE Multi-Tier Architecture

- Clear separation of presentation, business logic, and data layers.
- High scalability for enterprise applications.
- Easy maintenance and debugging.
- Better code reusability.
- Enhanced security.

- Supports distributed applications.
- Simplifies teamwork among developers.
- Efficient database access through JDBC.
- Suitable for large-scale enterprise systems.

1. J2EE Architecture Types

J2EE (Java 2 Platform, Enterprise Edition) — now known as **Jakarta EE** — is an enterprise computing platform designed to build large-scale, multi-tiered, scalable, and secure web applications.

Architectural Tiers

1. **2-Tier Architecture (Client-Server):**
    - **Client:** User interface (Web browser or thick application) communicates directly with the database.
    - **Server:** Database engine handles data storage and basic constraints.
    - *Drawback:* Poor scalability; business logic gets tightly coupled to either the UI or stored procedures.
2. **3-Tier Architecture:**
    - **Client Tier (Presentation):** Web browser renders HTML/CSS.
    - **Middle Tier (Application/Logic):** Web Server / App Server executes Servlets, JSPs, and business logic.
    - **Enterprise Information System (EIS) Tier (Data):** RDBMS (MySQL, Oracle) handles data persistence.
3. **N-Tier Architecture (Enterprise Model):**
    - **Client Tier:** Dynamic HTML browsers, Applets, or desktop GUI applications.
    - **Web Tier:** Web Container handling HTTP requests, Servlets, and JSP pages.
    - **Business Tier:** EJB (Enterprise Java Beans) Container managing transactions, security, and complex business workflows.
    - **EIS Tier:** Enterprise databases, legacy systems, and ERP software.

2. J2EE Containers

A **Container** is a runtime environment provided by an application server that manages the lifecycle, execution, security, threading, and resource injection for enterprise components.

Key J2EE Containers

2. J2EE Containers

Containers are runtime environments that provide services and manage the lifecycle of J2EE components. They are a fundamental part of the J2EE architecture.

- **Web Container (Servlet Container):**

    - Hosts and manages web components like Servlets, JSPs, and Filters.
    - Provides services such as request/response handling, session management, security, and lifecycle management for web components.
    - **Examples:** Apache Tomcat, Jetty, Undertow.
- **EJB Container (Enterprise JavaBean Container):**
    - Hosts and manages Enterprise JavaBeans (EJBs).
    - Provides services such as transaction management, security, concurrency, persistence, and lifecycle management for EJBs.
    - **Examples:** JBoss EAP, WebLogic, WebSphere.
- **Application Client Container (ACC):**
    - Hosts standalone J2EE application clients. These clients interact with EJBs and other J2EE components on the server.
    - Provides services for dependency injection, naming, and security for client applications.
- **Applet Container:**
    - Hosts Java Applets within a web browser. (Note: Applets are largely deprecated now due to security concerns and the rise of other web technologies).

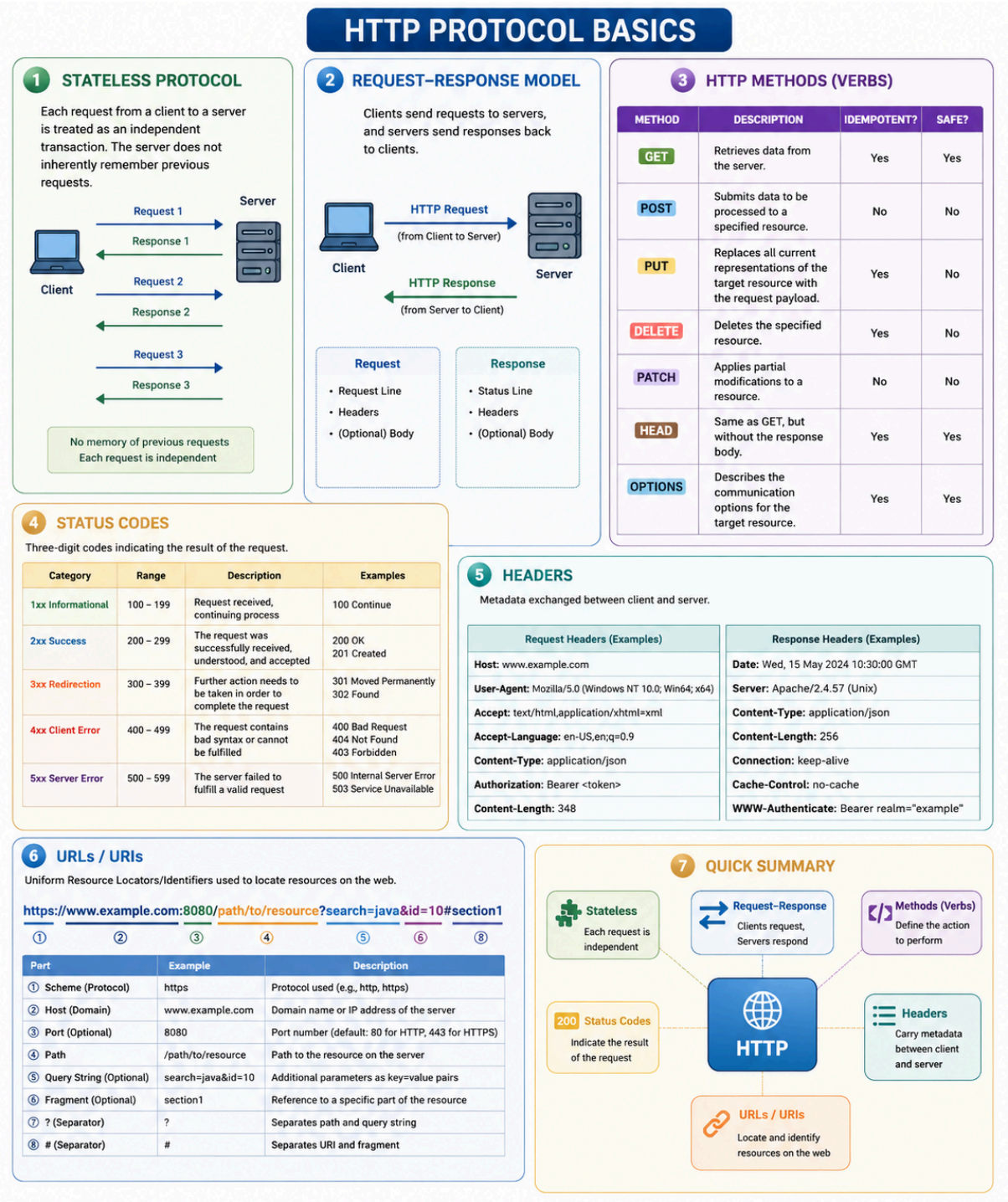
3. Types of Servers in J2EE Applications

Server Type	Primary Responsibility	Components Hosted	Key Examples
Web Server	Serves static content (HTML, CSS, JS) and delegates dynamic Java requests to a Web Container via HTTP.	HTML, Images, Servlets, JSPs	Apache HTTP Server, Nginx, Apache Tomcat
Application Server	Provides complete enterprise infrastructure including web containers, EJB containers, transaction management, and JMS messaging.	Servlets, JSPs, EJBs, Web Services, JMS	Red Hat WildFly (JBoss), Oracle WebLogic, IBM WebSphere
Database Server	Handles persistent data storage, ACID compliance, and query processing.	SQL Tables, Views, Stored Procedures	MySQL, PostgreSQL, Oracle DB

4. HTTP Protocols and API

HTTP (Hypertext Transfer Protocol) is the foundation of data communication for the World Wide Web.

- **HTTP Protocol Basics:**
    - **Stateless Protocol:** Each request from a client to a server is treated as an independent transaction. The server does not inherently remember previous requests.
    - **Request-Response Model:** Clients send requests to servers, and servers send responses back to clients.
    - **Methods (Verbs):**
        - GET: Retrieves data from the server. Idempotent and safe.
        - POST: Submits data to be processed to a specified resource. Not idempotent.
        - PUT: Replaces all current representations of the target resource with the request payload. Idempotent.
        - DELETE: Deletes the specified resource. Idempotent.
        - PATCH: Applies partial modifications to a resource. Not idempotent.
        - HEAD: Same as GET, but without the response body.
        - OPTIONS: Describes the communication options for the target resource.
    - **Status Codes:** Three-digit codes indicating the result of the request (e.g., 200 OK, 404 Not Found, 500 Internal Server Error).
    - **Headers:** Metadata exchanged between client and server (e.g., Content-Type, User-Agent, Authorization).
    - **URLs/URIs:** Uniform Resource Locators/Identifiers used to locate resources on the web.

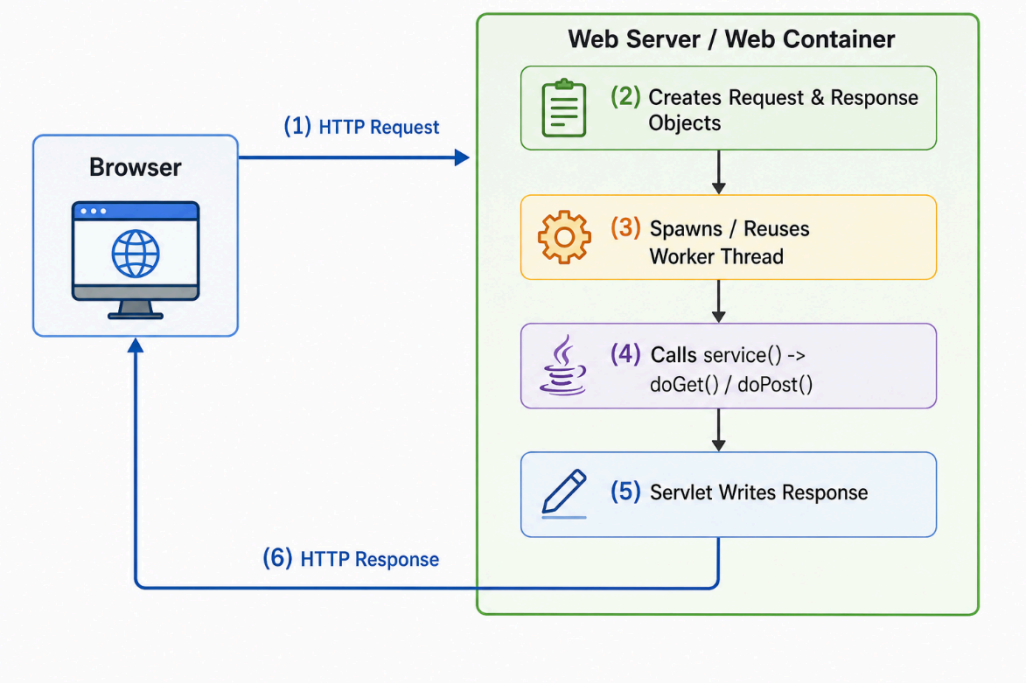


- **HTTP API (Servlet API - javax.servlet and javax.servlet.http):**
  - The core Java API for developing web components (Servlets and JSPs) that interact with HTTP requests and responses.
  - **HttpServletRequest Interface:**
    - Represents an HTTP request from the client to the server.
    - Provides methods to access:
      - Request parameters (getParameter(), getParameterValues(), getParameterMap())

        - Request headers (getHeader(), getHeaderNames())
        - Request attributes (getAttribute(), setAttribute())
        - Session information (getSession())
        - Request method (getMethod())
        - Request URI/URL (getRequestURI(), getRequestURL())
        - Input stream (getInputStream(), getReader())
        - Cookies (getCookies())
    - **HttpServletResponse Interface:**
        - Represents an HTTP response from the server to the client.
        - Provides methods to:
            - Set response headers (setHeader(), addHeader())
            - Set content type (setContentType())
            - Set status code (setStatus(), sendError())
            - Write response body (getWriter(), getOutputStream())
            - Redirect (sendRedirect())
            - Add cookies (addCookie())
    - **HttpSession Interface:**
        - Provides a way to maintain state (session data) between multiple requests from the same client.
        - Allows storing and retrieving attributes specific to a user's session.

5. Request Processing in Web Applications

Here is how a Servlet-based web request moves through a server step-by-step:



Execution Steps

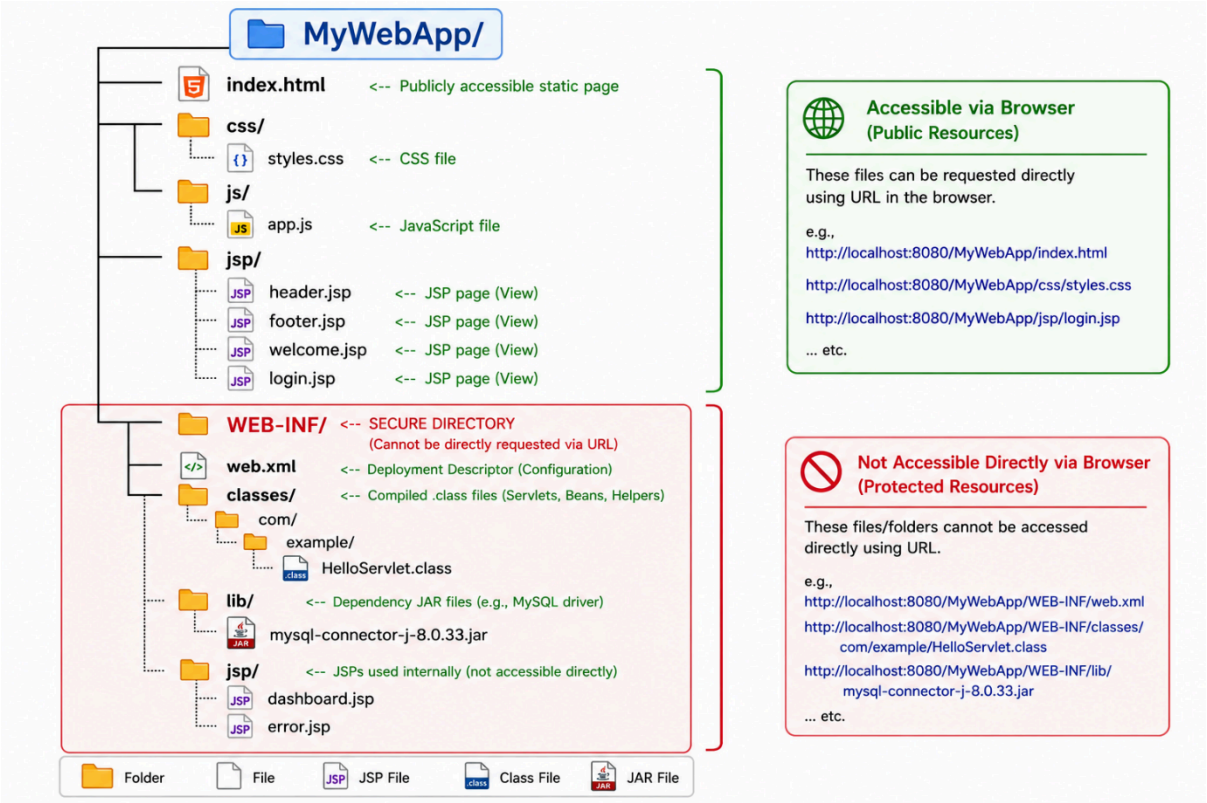
1. **Client Request:** The web browser sends an HTTP request to the web server/application server.
2. **Web Server/Application Server Receives Request:** The server receives the request and, if it's for dynamic content, forwards it to the appropriate Web Container.
3. **Web Container Locates Servlet/JSP:** Based on the URL pattern defined in the web application's deployment descriptor (web.xml) or annotations, the Web Container identifies the target Servlet or JSP.
4. **Servlet/JSP Processing:**
    - If it's a Servlet, the container calls its service() method, which in turn dispatches to doGet(), doPost(), etc., based on the HTTP method.
    - If it's a JSP, the container translates the JSP into a Servlet, compiles it, and then executes it.
5. **Accessing Request Data:** The Servlet/JSP uses the HttpServletRequest object to retrieve information about the incoming request (parameters, headers, session data).
6. **Business Logic Execution:** The Servlet/JSP often delegates to business logic components (e.g., service classes, DAOs, or even EJBs) to perform necessary operations (data validation, database interaction, calculations).
7. **Generating Response:** The Servlet/JSP uses the HttpServletResponse object to generate the HTTP response. This typically involves setting content type, headers, and writing the HTML (or other content) to the output stream.
8. **Forwarding/Redirection (Optional):**
    - **Forwarding:** The Servlet/JSP can forward the request to another resource (another Servlet, JSP, or static HTML) *internally* on the server. The URL in the browser does not change.

- **Redirection:** The Servlet/JSP can send a redirect instruction (HTTP 302 status code) to the browser, telling it to make a *new* request to a different URL. The URL in the browser *does* change.

9. **Response Sent to Client:** The Web Container sends the generated HTTP response back to the client.
10. **Client Renders Response:** The web browser receives the response and renders the content.

6. Web Application Structure (Standard Directory Layout)

Java Enterprise Web Applications follow a strict standardized directory hierarchy packaged into a **WAR (Web Application Archive)** file:



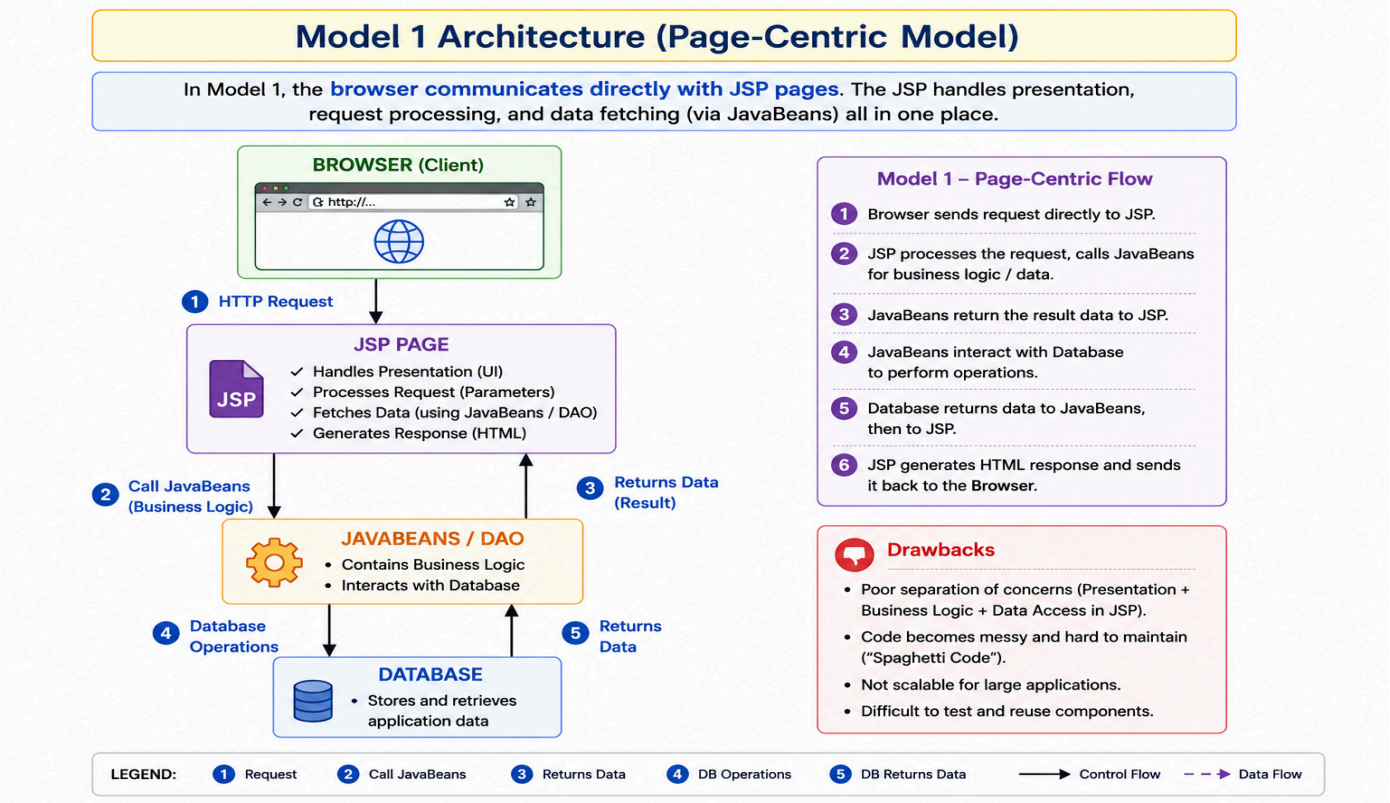
- **Root Directory:** Contains static web resources (HTML, CSS, JS, images) and JSP files.
- **WEB-INF/:** A special directory that is *not* directly accessible by clients. It contains:
    - **web.xml (Deployment Descriptor):** An XML file that configures the web application. It defines Servlets, Servlet mappings, filters, listeners, welcome files, error pages, security constraints, and more. While annotations have reduced its necessity for basic configurations, it's still crucial for complex setups or overriding defaults.
    - **classes/:** Contains compiled Java class files of the web application (Servlets, utility classes, JavaBeans, etc.).
    - **lib/:** Contains JAR files of third-party libraries that the web application depends on.

Note: Files located inside WEB-INF cannot be accessed directly by client browsers entering a URL. They are protected and accessible only via internal server dispatches (like RequestDispatcher).

7. Web Containers and Web Architecture Models

Architecture Models

Model 1 Architecture (Page-Centric Model)

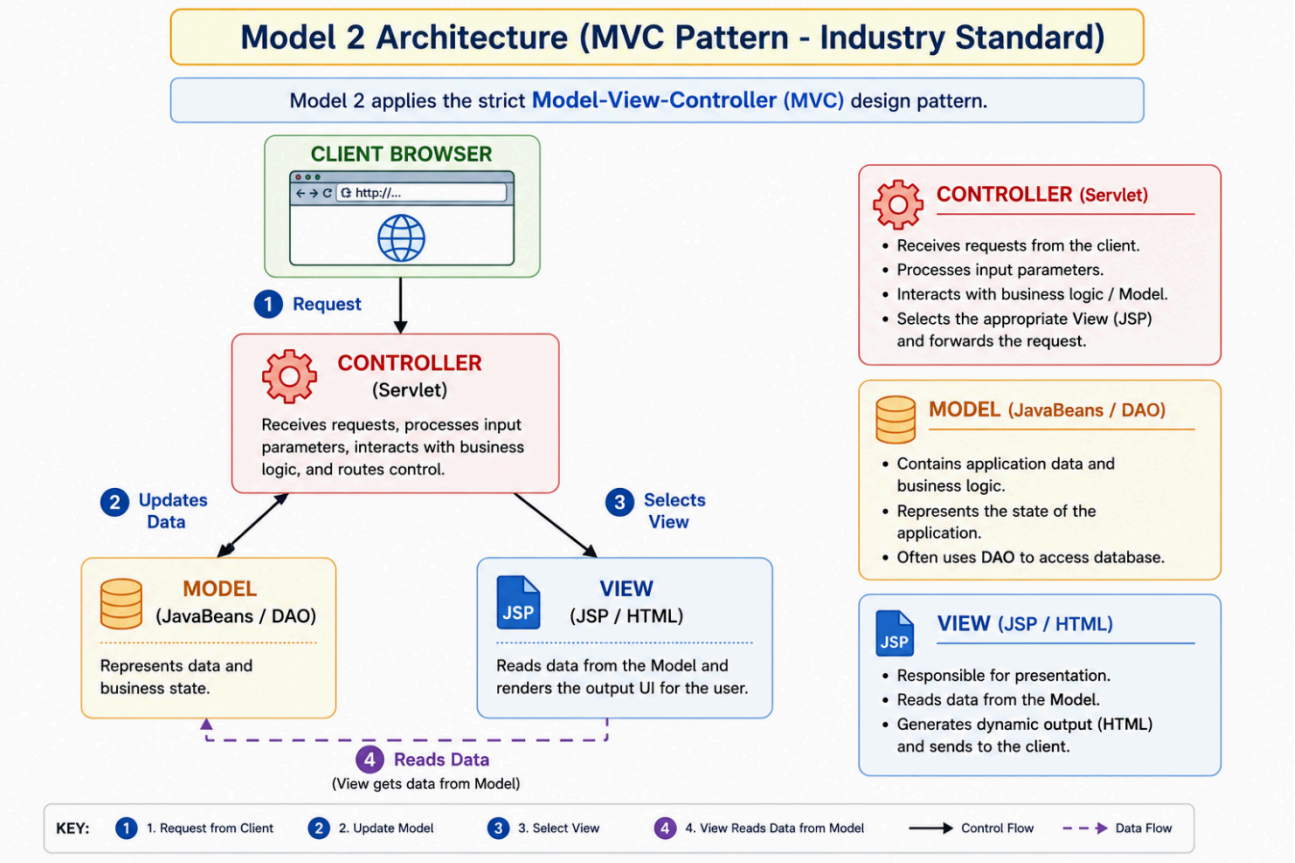


In Model 1, the browser communicates directly with JSP pages. The JSP handles presentation, request processing, and data fetching (via JavaBeans) all in one place.

- *Drawbacks:* Poor separation of concerns; code becomes messy and hard to maintain ("Spaghetti Code").

Model 2 Architecture (MVC Pattern - Industry Standard)

Model 2 applies the strict **Model-View-Controller (MVC)** design pattern:



- **Model-View-Controller (MVC):**
  - A widely adopted architectural pattern for separating concerns in web applications.
  - **Model:** Represents the data and business logic of the application. It's independent of the user interface.
  - **View:** Responsible for displaying the data to the user. In J2EE web apps, this is often JSPs or HTML templates.
  - **Controller:** Handles user input, interacts with the Model, and selects the appropriate View to display. In J2EE, Servlets or framework controllers (like in Spring MVC or Struts) act as controllers.
  - **Advantages:** Promotes separation of concerns, improves maintainability, enhances testability.
  - **How it applies to J2EE:** Servlets typically act as controllers, JavaBeans or POJOs (Plain Old Java Objects) as models, and JSPs as views. Frameworks like Spring MVC, Struts, and JSF provide robust MVC implementations on top of the Servlet API.

8. Practical Example Programs

Example A: Standard web.xml **(Deployment Descriptor)**

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        version="4.0">
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>com.example.HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

Example B: Hello World Servlet (Modern Annotation Based)

Java

```
package com.example;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

// @WebServlet Annotation replaces web.xml mapping in modern Servlet 3.0+
@WebServlet("/hello")
```

```
public class HelloServlet extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      // Write HTML output response
      PrintWriter out = response.getWriter();
      out.println("<html><body>");
      out.println("<h1>Hello from J2EE Web Container!</h1>");
      out.println("<p>Server Time: " + new java.util.Date() + "</p>");
      out.println("</body></html>");
   }
}
```

Example C: Request Processing & Form Data (Model 2 / MVC Servlet)

HTML Form (index.html**)**

HTML

```
<!DOCTYPE html>
<html>
<head><title>Login</title></head>
<body>
   <form action="login" method="POST">
      Username: <input type="text" name="username"><br><br>
      Password: <input type="password" name="password"><br><br>
      <input type="submit" value="Login">
   </form>
</body>
</html>
```

Controller Servlet (LoginServlet.java**)**

Java

```
package com.example;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/login")
public class LoginServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // Extract form request parameters
        String user = request.getParameter("username");
        String pass = request.getParameter("password");

        // Simple validation logic
        if ("admin".equals(user) && "secret123".equals(pass)) {
            // Attach data to request scope
            request.setAttribute("user", user);

            // Forward control to success view
            RequestDispatcher dispatcher = request.getRequestDispatcher("welcome.jsp");
            dispatcher.forward(request, response);
```

```
        } else {
            // Redirect back to error page
            response.sendRedirect("error.html");
        }
    }
}
```