Comprehensive Java EE Servlets & JSP Learning Guide

Table of Contents

1. Java EE Fundamentals

1.1 The Java EE Architecture and Evolution

What is Java EE?

Java Enterprise Edition (Java EE), now known as Jakarta EE, is a set of specifications that extends Java Standard Edition (Java SE) with enterprise features like web services, distributed computing, and web applications. It provides a platform for developers to build robust, scalable, and secure enterprise applications.

Evolution of Java EE

1. **J2EE (Java 2 Enterprise Edition)**: The initial releases focused on providing enterprise capabilities.
2. **Java EE 5**: Introduced annotations, simplified development, and improved EJB 3.0.
3. **Java EE 6**: Added more annotations, Servlet 3.0 with asynchronous capabilities, and CDI (Contexts and Dependency Injection).
4. **Java EE 7**: Enhanced HTML5 support, WebSocket API, JSON processing.
5. **Java EE 8**: Updated Servlet 4.0 with HTTP/2 support, enhanced JSON processing, new Security API.
6. **Jakarta EE** (after Oracle transferred Java EE to the Eclipse Foundation): Continues the evolution with backward compatibility.

Java EE Architecture

Java EE follows a multi-tier architecture:

1. **Client Tier**: Web browsers, mobile apps, desktop applications
2. **Web Tier**: Servlets, JSP, JSF (Java Server Faces)
3. **Business Tier**: EJB (Enterprise JavaBeans), CDI beans
4. **Data Tier**: JPA (Java Persistence API), JDBC

![Java EE Architecture]

The **Web Tier** is where Servlets and JSP reside, providing the interface between clients and the business logic. This guide focuses primarily on this tier.

Key Principles of Java EE

1. **Component-based**: Applications are composed of reusable components.
2. **Declarative Security**: Security configuration is separate from business logic.
3. **Container Services**: Application servers provide services like security, transaction management, and lifecycle management.
4. **Configuration over Convention**: XML configuration files and annotations define application behavior.

Java EE was designed with these principles to create maintainable, scalable enterprise applications.

1.2 Web Containers and Application Servers

Web Containers vs. Application Servers

Web Containers (or Servlet Containers) implement the Servlet and JSP specifications, while **Application Servers** implement the full Java EE specification.

Feature	Web Container (e.g., Tomcat)	Application Server (e.g., WildFly)
Servlet/JSP Support	✓	✓
EJB Support	✗	✓
JPA Support	Limited	✓
JMS Support	✗	✓
Full Java EE/Jakarta EE	✗	✓

Popular Web Containers and Application Servers

1. **Apache Tomcat**
    - A popular, lightweight web container
    - Implements Servlet, JSP, WebSocket specifications
    - Excellent for simpler web applications
2. **Jetty**
    - Lightweight, embeddable web container
    - Often used in embedded scenarios and microservices
3. **WildFly** (formerly JBoss AS)
    - Full Java EE application server
    - Open-source, backed by Red Hat
    - Supports clustering and high availability
4. **GlassFish**
    - Reference implementation of Java EE
    - Now maintained by the Eclipse Foundation as part of Jakarta EE
5. **IBM WebSphere**
    - Commercial application server
    - Focuses on enterprise-grade reliability
6. **Oracle WebLogic**
    - Commercial application server
    - Strong support for Oracle databases

Choosing the Right Container

- **Use a Web Container** when you only need Servlet/JSP capabilities and want a lightweight solution.
- **Use an Application Server** when you need EJB, JMS, full JPA support, or other Java EE features.

For learning purposes and simpler applications, Apache Tomcat is an excellent choice.

1.3 Setting Up a Development Environment

Required Software

1. **JDK (Java Development Kit)**: Version 8 or newer
2. **Apache Tomcat**: Version 9.x or 10.x (note that Tomcat 10.x uses Jakarta EE namespace)
3. **IDE**: Eclipse IDE for Enterprise Java Developers, IntelliJ IDEA Ultimate, or NetBeans
4. **Build Tool**: Maven or Gradle (optional but recommended)

Step-by-Step Setup with Eclipse and Tomcat

1. **Install JDK**
    - Download from Oracle or use OpenJDK
    - Set JAVA_HOME environment variable
2. **Install Apache Tomcat**
    - Download from Tomcat website
    - Extract to a directory (e.g., `C:\tomcat` or `/opt/tomcat`)
    - Set CATALINA_HOME environment variable to this directory
3. **Install Eclipse IDE for Enterprise Java Developers**
    - Download from Eclipse website
    - Extract and run the installer
4. **Configure Tomcat in Eclipse**
    - Open Eclipse
    - Go to Window > Preferences > Server > Runtime Environments
    - Click "Add"
    - Select Apache Tomcat v9.0 (or your version)
    - Browse to your Tomcat installation directory
    - Click Finish
5. **Create a Server in Eclipse**
    - Go to bottom panel, Servers tab (if not visible, Window > Show View > Servers)
    - Click "New server wizard" link
    - Select "Tomcat v9.0 Server"
    - Click Next and Finish
6. **Configure Maven (Optional but Recommended)**
    - Eclipse typically includes Maven integration
    - Go to Window > Preferences > Maven to configure settings

Alternative: IntelliJ IDEA Setup

1. **Install IntelliJ IDEA Ultimate**
    - Download from JetBrains website
    - Run the installer
2. **Configure Tomcat in IntelliJ**
    - Open IntelliJ IDEA
    - Go to File > Settings > Build, Execution, Deployment > Application Servers
    - Click + and select Tomcat Server
    - Browse to your Tomcat installation directory
    - Click OK
3. **Create a Run Configuration**
    - Go to Run > Edit Configurations
    - Click + and select Tomcat Server > Local
    - Configure server settings and deployment options
    - Click OK

Creating Your First Java Web Project

In Eclipse:

1. File > New > Dynamic Web Project
2. Enter project name (e.g., "FirstWebApp")
3. Select Target Runtime: Apache Tomcat v9.0
4. Click Next, configure source folders
5. Check "Generate web.xml deployment descriptor"
6. Click Finish

In IntelliJ IDEA:

1. File > New > Project
2. Select Java Enterprise
3. Check "Web Application"
4. Select Application Server: Tomcat
5. Click Next, configure project name and location
6. Click Finish

Maven-based Setup (Recommended for Production)

Create a Maven project with the following `pom.xml`:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>com.example</groupId>
    <artifactId>first-web-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <!-- For Tomcat 9.x (Java EE 8) -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.3</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- For Tomcat 10.x (Jakarta EE 9+), use these instead:
        <dependency>
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <version>5.0.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jakarta.servlet.jsp</groupId>
            <artifactId>jakarta.servlet.jsp-api</artifactId>
            <version>3.0.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jakarta.servlet.jsp.jstl</groupId>
            <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
            <version>2.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.web</groupId>
            <artifactId>jakarta.servlet.jsp.jstl</artifactId>
            <version>2.0.0</version>
        </dependency>
```

```
        -->
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.2</version>
            </plugin>
        </plugins>
    </build>
</project>
```

1.4 Web Application Structure

Standard Directory Structure

A Java web application follows a specific directory structure:

```
WebApp/
│
├── src/
│   └── main/
│       ├── java/        # Java source files
│       ├── resources/   # Properties, configurations
│       └── webapp/      # Web resources
│           ├── WEB-INF/
│           │   ├── web.xml           # Deployment descriptor
│           │   ├── classes/          # Compiled Java classes
│           │   ├── lib/              # JAR dependencies
│           │   └── tags/             # Custom JSP tags
│           │
│           ├── META-INF/             # Application metadata
│           ├── resources/            # Static resources
│           │   ├── css/              # Stylesheets
│           │   ├── js/               # JavaScript files
│           │   └── images/           # Image files
│           │
│           └── *.jsp                 # JSP pages
│
└── pom.xml              # Maven build file (if using Maven)
```

Key Components

1. **WEB-INF Directory**:
    - Not accessible directly by the client
    - Contains web.xml, classes, and libraries

- Protected from direct access

2. **web.xml** (Deployment Descriptor):
   - Configures how the web application is deployed
   - Defines servlets, filters, listeners
   - Maps URLs to servlets
   - Newer versions of Java EE allow annotations to replace much of web.xml
3. **Classes Directory**:
   - Contains compiled Java classes
   - Automatically included in the classpath
4. **Lib Directory**:
   - Contains JAR dependencies
   - Automatically included in the classpath
5. **Public Resources**:
   - Everything outside WEB-INF is publicly accessible
   - HTML, JSP, CSS, JavaScript, images

Sample web.xml (Deployment Descriptor)

```
<?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">

    <display-name>My First Web Application</display-name>

    <!-- Servlet Definition -->
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>com.example.HelloServlet</servlet-class>
        <init-param>
            <param-name>greeting</param-name>
            <param-value>Welcome to Java EE!</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
```

```
    <!-- Welcome File List -->
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <!-- Session Configuration -->
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

    <!-- Error Pages -->
    <error-page>
        <error-code>404</error-code>
        <location>/error404.jsp</location>
    </error-page>

    <!-- Context Parameters -->
    <context-param>
        <param-name>dbURL</param-name>
        <param-value>jdbc:mysql://localhost:3306/mydb</param-value>
    </context-param>
</web-app>
```

Web Application Deployment Descriptor Versions

Servlet Spec	Java EE/Jakarta EE Version	web.xml Root Element
2.5	Java EE 5	`<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">`
3.0	Java EE 6	`<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">`
3.1	Java EE 7	`<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">`
4.0	Java EE 8	`<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="4.0">`
5.0	Jakarta EE 9	`<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee" version="5.0">`

1.5 Building and Deploying Web Applications

Building Web Applications

Manual Build:

1. Compile Java classes
2. Place compiled classes in WEB-INF/classes

3. Place JAR files in WEB-INF/lib
4. Create WAR file (Web Application Archive)

Maven Build:

```
mvn clean package
```

Gradle Build:

```
gradle clean build
```

Deployment Methods

1. **IDE Deployment** (Development):
    - Right-click on project > Run As > Run on Server
    - IDE handles the build and deployment
2. **Manual Deployment** (WAR file):
    - Copy WAR file to server's webapps directory
    - Tomcat automatically deploys when started
3. **Tomcat Manager Application**:
    - Access http://localhost:8080/manager/html
    - Upload WAR file through web interface
    - Configure users in tomcat-users.xml:

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="admin" password="password" roles="manager-gui"/>
</tomcat-users>
```

4. **Programmatic Deployment**:
    - Use Cargo or other deployment tools
    - Integrate with CI/CD pipelines

Deployment with Maven

Add to pom.xml:

```
<build>
    <plugins>
```

```
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <url>http://localhost:8080/manager/text</url>
                    <server>TomcatServer</server>
                    <path>/myapp</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

Add to Maven settings.xml:

```
<servers>
    <server>
        <id>TomcatServer</id>
        <username>admin</username>
        <password>password</password>
    </server>
</servers>
```

Deploy with:

```
mvn tomcat7:deploy
```

Hot Deployment vs. Cold Deployment

Hot Deployment:

- Deploys without restarting the server
- Faster for development
- May cause memory leaks if done repeatedly

Cold Deployment:

- Requires server restart
- More reliable for production
- Cleans up resources properly

Tomcat Configuration Files

1. **server.xml**:
    - Main configuration file
    - Defines connectors, hosts, and services

- Located in CATALINA_HOME/conf

2. **context.xml**:
   - Configures the default context for all applications
   - Can be overridden by application-specific context files
   - Located in CATALINA_HOME/conf
3. **web.xml**:
   - Default web.xml for all applications
   - Located in CATALINA_HOME/conf
4. **tomcat-users.xml**:
   - Defines users, roles, and permissions
   - Located in CATALINA_HOME/conf

Common Deployment Issues and Solutions

1. **ClassNotFoundException**:
   - Ensure all required JARs are in WEB-INF/lib
   - Check class path configuration
2. **Context Path Conflicts**:
   - Each application must have a unique context path
   - Check server.xml for duplicate paths
3. **Port Conflicts**:
   - Change port in server.xml if 8080 is already in use
   - Common alternative ports: 8081, 8082, 8888
4. **Permission Issues**:
   - Ensure proper file permissions on the server
   - Check user permissions for deployment
5. **Memory Issues**:
   - Increase JVM heap size with -Xmx and -Xms options
   - Set in CATALINA_OPTS environment variable

2. Servlets

2.1 Servlet Lifecycle and Architecture

What is a Servlet?

A Servlet is a Java class that handles requests and generates responses in a web application. Servlets are managed by a web container (like Tomcat) and follow a specific lifecycle.

Servlet Architecture

Servlets are part of the Java EE's web tier and implement the request-response model:

1. Client sends a request to the web server
2. Web server passes the request to the servlet container
3. Servlet container creates request and response objects
4. Servlet container calls the appropriate servlet method
5. Servlet processes the request and writes to the response
6. Servlet container sends the response back to the client

Servlet Lifecycle

The servlet lifecycle has four main phases:

1. **Loading and Instantiation**:
    - Container loads the servlet class
    - Container creates an instance of the servlet
2. **Initialization**:
    - Container calls the `init(ServletConfig)` method
    - Occurs only once in the servlet's lifecycle
    - Used for one-time setup (DB connections, resources)
3. **Request Handling**:
    - Container calls `service()` method for each request
    - `service()` typically dispatches to `doGet()`, `doPost()`, etc.
    - Multiple threads may call `service()` concurrently
4. **Destruction**:
    - Container calls the `destroy()` method
    - Occurs when the servlet is unloaded
    - Used for cleanup (closing connections, releasing resources)

Servlet Lifecycle Methods

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

@WebServlet("/lifecycle")
public class LifecycleServlet extends HttpServlet {

    // Called once when servlet is first loaded
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
```

```
        System.out.println("Servlet " + this.getServletName() + " has been
initialized");
        // Initialization code (DB connections, resource allocation)
    }

    // Called for each HTTP GET request
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        System.out.println("Servlet handling GET request");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Servlet Lifecycle Demonstration</h2>");
        out.println("<p>The servlet is now handling a GET request.</p>");
        out.println("</body></html>");
    }

    // Called for each HTTP POST request
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        // Handle POST requests
    }

    // Called when servlet is being unloaded
    public void destroy() {
        System.out.println("Servlet " + this.getServletName() + " is being
destroyed");
        // Cleanup code (close DB connections, release resources)
    }
}
```

Servlet Configuration

Servlets can be configured either through the deployment descriptor (web.xml) or using annotations (Servlet 3.0+).

web.xml configuration:

```
<servlet>
    <servlet-name>LifecycleServlet</servlet-name>
    <servlet-class>com.example.LifecycleServlet</servlet-class>
    <init-param>
        <param-name>database</param-name>
        <param-value>jdbc:mysql://localhost/mydb</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>LifecycleServlet</servlet-name>
```

```
    <url-pattern>/lifecycle</url-pattern>
</servlet-mapping>
```

Annotation configuration (Servlet 3.0+):

```
@WebServlet(
    name = "LifecycleServlet",
    urlPatterns = {"/lifecycle", "/lifecycle/*"},
    initParams = {
        @WebInitParam(name = "database", value = "jdbc:mysql://localhost/mydb")
    },
    loadOnStartup = 1
)
public class LifecycleServlet extends HttpServlet {
    // Servlet implementation
}
```

Single-Thread Model vs. Multi-Thread Model

By default, servlets use a multi-threaded model:

- One servlet instance handles multiple requests concurrently
- Each request is processed in a separate thread
- Servlet code must be thread-safe

For thread safety:

- Avoid instance variables (or make them thread-safe)
- Use thread-local storage for thread-specific data
- Synchronize access to shared resources

The single-thread model (implementing `SingleThreadModel` interface) is deprecated.

2.2 HttpServlet, GenericServlet, and Servlet Interfaces

Servlet Interface Hierarchy

Java servlets follow a hierarchical interface structure:

1. **Servlet Interface** (javax.servlet.Servlet)
    - The base interface that all servlets must implement
    - Defines basic lifecycle methods (init, service, destroy)
2. **GenericServlet Class** (javax.servlet.GenericServlet)
    - Abstract class implementing Servlet interface
    - Protocol-independent implementation
    - Provides convenience methods for parameter access

3. **HttpServlet Class** (javax.servlet.http.HttpServlet)
    - Extends GenericServlet
    - Specialized for HTTP protocol
    - Defines HTTP-specific methods (doGet, doPost, etc.)

Servlet Interface

```
public interface Servlet {
    public void init(ServletConfig config) throws ServletException;
    public ServletConfig getServletConfig();
    public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException;
    public String getServletInfo();
    public void destroy();
}
```

The Servlet interface is the foundation of all servlets. It defines the basic lifecycle methods that a servlet container calls.

GenericServlet Class

```
public abstract class GenericServlet implements Servlet, ServletConfig,
Serializable {
    private ServletConfig config;

    public void init(ServletConfig config) throws ServletException {
        this.config = config;
        this.init();
    }

    public void init() throws ServletException {
        // Subclasses override this method for initialization
    }

    public ServletConfig getServletConfig() {
        return config;
    }

    public abstract void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException;

    public String getServletInfo() {
        return "";
    }

    public void destroy() {
        // Subclasses override this method for cleanup
    }
```

```
    // Convenience methods for parameter access
    public String getInitParameter(String name) {
        return config.getInitParameter(name);
    }

    public Enumeration<String> getInitParameterNames() {
        return config.getInitParameterNames();
    }

    public ServletContext getServletContext() {
        return config.getServletContext();
    }

    public String getServletName() {
        return config.getServletName();
    }
}
```

GenericServlet provides a protocol-independent implementation of the Servlet interface. It's useful for creating servlets that aren't specific to HTTP.

HttpServlet Class

```
public abstract class HttpServlet extends GenericServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // Default implementation returns a 405 Method Not Allowed error
        resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED);
    }

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // Default implementation returns a 405 Method Not Allowed error
        resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED);
    }

    protected void doPut(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // Default implementation returns a 405 Method Not Allowed error
        resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED);
    }

    protected void doDelete(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // Default implementation returns a 405 Method Not Allowed error
        resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED);
    }

    protected void doHead(HttpServletRequest req, HttpServletResponse resp)
```

```
            throws ServletException, IOException {
        // Default implementation calls doGet but doesn't return the body
    }

    protected void doOptions(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // Default implementation sets the Allow header with supported HTTP
methods
    }

    protected void doTrace(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // Default implementation returns a 405 Method Not Allowed error
        resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED);
    }

    protected void service(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // Dispatches requests to the appropriate do* method based on HTTP method
        String method = req.getMethod();
        if (method.equals("GET")) {
            doGet(req, resp);
        } else if (method.equals("POST")) {
            doPost(req, resp);
        } else if (method.equals("PUT")) {
            doPut(req, resp);
        } else if (method.equals("DELETE")) {
            doDelete(req, resp);
        } else if (method.equals("HEAD")) {
            doHead(req, resp);
        } else if (method.equals("OPTIONS")) {
            doOptions(req, resp);
        } else if (method.equals("TRACE")) {
            doTrace(req, resp);
        } else {
            resp.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED);
        }
    }

    public void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
        // Casts the request and response to HTTP types and calls the HTTP-
specific service method
        HttpServletRequest request = (HttpServletRequest) req;
        HttpServletResponse response = (HttpServletResponse) res;
        service(request, response);
    }
}
```

HttpServlet is specialized for HTTP protocols and is the most commonly used servlet base class. It provides HTTP-specific methods like doGet() and doPost().

Implementing a Basic HttpServlet

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

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

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello from Servlet!</h1>");
        out.println("<p>The time now is: " + new java.util.Date() + "</p>");
        out.println("</body>");
        out.println("</html>");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        String name = request.getParameter("name");

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello, " + name + "!</h1>");
        out.println("<p>Thank you for submitting the form.</p>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

When to Use Each Type

- **Servlet Interface**: Rarely implemented directly; use it when you need complete control over the servlet lifecycle.
- **GenericServlet**: Use when creating protocol-independent servlets (uncommon).
- **HttpServlet**: Use for most web applications where HTTP is the protocol.

2.3 Request and Response Handling

HttpServletRequest

The `HttpServletRequest` interface extends `ServletRequest` and provides HTTP-specific methods for accessing request data.

Key Methods:

```
// Getting basic request information
String getMethod();             // Returns the HTTP method (GET, POST, etc.)
String getRequestURI();         // Returns the URI part of the URL
String getQueryString();        // Returns the query string
String getContextPath();        // Returns the context path (application path)
String getServletPath();        // Returns the servlet path within the application

// Working with parameters
String getParameter(String name);                  // Returns a parameter value
Map<String, String[]> getParameterMap();           // Returns all parameters
Enumeration<String> getParameterNames();           // Returns all parameter names
String[] getParameterValues(String name);          // Returns all values for a
parameter

// Working with headers
String getHeader(String name);                     // Returns a header value
Enumeration<String> getHeaderNames();              // Returns all header names
Enumeration<String> getHeaders(String name);       // Returns all values for a
header

// Working with cookies
Cookie[] getCookies();                             // Returns all cookies

// Working with sessions
HttpSession getSession();                          // Gets or creates a session
HttpSession getSession(boolean create);            // Gets or creates a session
based on parameter

// Working with request attributes
Object getAttribute(String name);                  // Gets an attribute
void setAttribute(String name, Object value);      // Sets an attribute
Enumeration<String> getAttributeNames();           // Returns all attribute names
void removeAttribute(String name);                 // Removes an attribute

// Working with the request body
BufferedReader getReader();                        // Gets a reader for text data
ServletInputStream getInputStream();               // Gets an input stream for
binary data
```

```
// Special attributes
RequestDispatcher getRequestDispatcher(String path);  // Gets a dispatcher for
forwarding
```

Examples of Using HttpServletRequest:

```
@WebServlet("/request-demo")
public class RequestDemoServlet extends HttpServlet {

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

        // Basic request information
        String method = request.getMethod();
        String uri = request.getRequestURI();
        String queryString = request.getQueryString();
        String contextPath = request.getContextPath();
        String servletPath = request.getServletPath();

        // Working with parameters
        String name = request.getParameter("name");
        String[] hobbies = request.getParameterValues("hobby");

        // Working with headers
        String userAgent = request.getHeader("User-Agent");
        Enumeration<String> headerNames = request.getHeaderNames();

        // Working with cookies
        Cookie[] cookies = request.getCookies();

        // Working with the request body
        // (for POST requests typically)
        // BufferedReader reader = request.getReader();
        // String requestBody = reader.lines().collect(Collectors.joining());

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>Request Demo</title></head>");
        out.println("<body>");
        out.println("<h1>Request Information</h1>");

        out.println("<h2>Basic Request Info</h2>");
        out.println("<ul>");
        out.println("<li>Method: " + method + "</li>");
        out.println("<li>URI: " + uri + "</li>");
        out.println("<li>Query String: " + queryString + "</li>");
```

```
            out.println("<li>Context Path: " + contextPath + "</li>");
            out.println("<li>Servlet Path: " + servletPath + "</li>");
            out.println("</ul>");

            out.println("<h2>Parameters</h2>");
            out.println("<ul>");
            out.println("<li>Name: " + name + "</li>");
            if (hobbies != null) {
                out.println("<li>Hobbies: " + String.join(", ", hobbies) + "</li>");
            }
            out.println("</ul>");

            out.println("<h2>Headers</h2>");
            out.println("<ul>");
            out.println("<li>User-Agent: " + userAgent + "</li>");
            while (headerNames.hasMoreElements()) {
                String headerName = headerNames.nextElement();
                out.println("<li>" + headerName + ": " + request.getHeader(headerName)
+ "</li>");
            }
            out.println("</ul>");

            out.println("<h2>Cookies</h2>");
            out.println("<ul>");
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    out.println("<li>" + cookie.getName() + ": " + cookie.getValue() +
"</li>");
                }
            } else {
                out.println("<li>No cookies found</li>");
            }
            out.println("</ul>");

            out.println("</body>");
            out.println("</html>");
        }
    }
```

HttpServletResponse

The `HttpServletResponse` interface extends `ServletResponse` and provides HTTP-specific methods for creating responses.

Key Methods:

```
// Setting response headers
void setHeader(String name, String value);     // Sets a header
void addHeader(String name, String value);     // Adds a value to a header
void setIntHeader(String name, int value);     // Sets an integer header
void addIntHeader(String name, int value);     // Adds an integer value to a
```

```
header
void setDateHeader(String name, long date);    // Sets a date header
void addDateHeader(String name, long date);    // Adds a date value to a header

// Setting response status
void setStatus(int sc);                        // Sets the status code
void sendError(int sc) throws IOException;     // Sends an error status
void sendError(int sc, String msg) throws IOException;  // Sends an error status
with message
void sendRedirect(String location) throws IOException;  // Redirects to another
URL

// Working with cookies
void addCookie(Cookie cookie);                 // Adds a cookie to the response

// Setting content type and properties
void setContentType(String type);              // Sets the content type
void setCharacterEncoding(String charset);     // Sets the character encoding
void setContentLength(int len);                // Sets the content length
void setBufferSize(int size);                  // Sets the buffer size
void flushBuffer() throws IOException;         // Flushes the buffer
void reset();                                  // Resets the response

// Getting output streams
PrintWriter getWriter() throws IOException;     // Gets a writer for text output
ServletOutputStream getOutputStream() throws IOException;  // Gets an output
stream for binary output
```

Examples of Using HttpServletResponse:

```
@WebServlet("/response-demo")
public class ResponseDemoServlet extends HttpServlet {

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

        // Get the action parameter to demonstrate different response features
        String action = request.getParameter("action");

        if ("redirect".equals(action)) {
            // Demonstrates redirect
            response.sendRedirect("https://www.google.com");
            return;
        } else if ("error".equals(action)) {
            // Demonstrates error handling
            response.sendError(HttpServletResponse.SC_NOT_FOUND, "The requested
resource was not found");
            return;
        } else if ("json".equals(action)) {
            // Demonstrates JSON response
            response.setContentType("application/json");
```

```
            response.setCharacterEncoding("UTF-8");

            PrintWriter out = response.getWriter();
            out.print("{\"message\": \"Hello, World!\", \"timestamp\": " +
System.currentTimeMillis() + "}");
            return;
        } else if ("image".equals(action)) {
            // Demonstrates binary response (a simple red dot image)
            response.setContentType("image/png");

            // Create a simple PNG image (a red dot)
            BufferedImage image = new BufferedImage(100, 100,
BufferedImage.TYPE_INT_RGB);
            Graphics2D g2 = image.createGraphics();
            g2.setColor(Color.RED);
            g2.fillOval(25, 25, 50, 50);
            g2.dispose();

            // Write the image to the response output stream
            ServletOutputStream out = response.getOutputStream();
            ImageIO.write(image, "png", out);
            return;
        } else if ("cookie".equals(action)) {
            // Demonstrates setting a cookie
            Cookie cookie = new Cookie("username", "JohnDoe");
            cookie.setMaxAge(60 * 60 * 24); // 1 day
            response.addCookie(cookie);

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.println("<html><body>");
            out.println("<h1>Cookie Set</h1>");
            out.println("<p>A cookie named 'username' has been set.</p>");
            out.println("</body></html>");
            return;
        }

        // Default: demonstrate a basic HTML response
        response.setContentType("text/html");
        response.setHeader("Cache-Control", "no-cache, no-store, must-
revalidate");

        PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>Response Demo</title></head>");
        out.println("<body>");
        out.println("<h1>Response Demonstration</h1>");

        out.println("<ul>");
        out.println("<li><a href='?action=redirect'>Redirect to Google</a></li>");
        out.println("<li><a href='?action=error'>Show Error Page</a></li>");
        out.println("<li><a href='?action=json'>Show JSON Response</a></li>");
```

```
        out.println("<li><a href='?action=image'>Show Image Response</a></li>");
        out.println("<li><a href='?action=cookie'>Set a Cookie</a></li>");
        out.println("</ul>");

        out.println("</body>");
        out.println("</html>");
    }
}
```

Handling Form Submissions

HTML Form Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>User Registration</title>
  </head>
  <body>
    <h1>User Registration</h1>

    <form action="register" method="post">
      <div>
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required />
      </div>
      <div>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required />
      </div>
      <div>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required />
      </div>
      <div>
        <label>Interests:</label>
        <input type="checkbox" id="tech" name="interests" value="technology" />
        <label for="tech">Technology</label>
        <input type="checkbox" id="sports" name="interests" value="sports" />
        <label for="sports">Sports</label>
        <input type="checkbox" id="music" name="interests" value="music" />
        <label for="music">Music</label>
      </div>
      <div>
        <label for="country">Country:</label>
        <select id="country" name="country">
          <option value="us">United States</option>
          <option value="ca">Canada</option>
          <option value="uk">United Kingdom</option>
          <option value="au">Australia</option>
        </select>
```

```
      </div>
      <div>
        <input type="submit" value="Register" />
      </div>
    </form>
  </body>
</html>
```

Servlet Handling the Form:

```
@WebServlet("/register")
public class RegistrationServlet extends HttpServlet {

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

        // Retrieve form parameters
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        String[] interests = request.getParameterValues("interests");
        String country = request.getParameter("country");

        // Validate inputs (simplified)
        if (name == null || name.trim().isEmpty() ||
            email == null || email.trim().isEmpty() ||
            password == null || password.trim().isEmpty()) {

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.println("<html><body>");
            out.println("<h1>Error</h1>");
            out.println("<p>All fields are required.</p>");
            out.println("<a href='javascript:history.back()'>Go Back</a>");
            out.println("</body></html>");
            return;
        }

        // Process the registration (in a real app, store in database)
        // For demonstration, just display the entered information

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>Registration Successful</title></head>");
        out.println("<body>");
        out.println("<h1>Registration Successful</h1>");
```

```
        out.println("<h2>Your Information:</h2>");
        out.println("<ul>");
        out.println("<li>Name: " + name + "</li>");
        out.println("<li>Email: " + email + "</li>");
        out.println("<li>Password: " + "*".repeat(password.length()) + "</li>");

        out.println("<li>Interests: ");
        if (interests != null && interests.length > 0) {
            out.println(String.join(", ", interests));
        } else {
            out.println("None selected");
        }
        out.println("</li>");

        out.println("<li>Country: " + getCountryName(country) + "</li>");
        out.println("</ul>");

        out.println("</body>");
        out.println("</html>");
    }

    // Helper method to convert country code to name
    private String getCountryName(String countryCode) {
        switch (countryCode) {
            case "us": return "United States";
            case "ca": return "Canada";
            case "uk": return "United Kingdom";
            case "au": return "Australia";
            default: return countryCode;
        }
    }
}
```

Handling File Uploads

For file uploads, you need to add `enctype="multipart/form-data"` to your form and use the `Part` interface from Servlet 3.0+, or a library like Apache Commons FileUpload for earlier versions.

HTML Form with File Upload:

```
<!DOCTYPE html>
<html>
  <head>
    <title>File Upload</title>
  </head>
  <body>
    <h1>File Upload</h1>

    <form action="upload" method="post" enctype="multipart/form-data">
      <div>
        <label for="file">Choose a file:</label>
```

```
        <input type="file" id="file" name="file" required />
      </div>
      <div>
        <label for="description">Description:</label>
        <input type="text" id="description" name="description" />
      </div>
      <div>
        <input type="submit" value="Upload" />
      </div>
    </form>
  </body>
</html>
```

Servlet Handling File Upload (Servlet 3.0+):

```
@WebServlet("/upload")
@MultipartConfig(
    fileSizeThreshold = 1024 * 1024,      // 1 MB
    maxFileSize = 1024 * 1024 * 10,       // 10 MB
    maxRequestSize = 1024 * 1024 * 100    // 100 MB
)
public class FileUploadServlet extends HttpServlet {

    // Define a location to store uploaded files
    private static final String UPLOAD_DIRECTORY = "uploads";

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

        // Gets the file part from the request
        Part filePart = request.getPart("file");
        String description = request.getParameter("description");

        // Extracts file name from content-disposition header
        String fileName = getSubmittedFileName(filePart);

        // Creates upload folder if it doesn't exist
        String uploadPath = getServletContext().getRealPath("") + File.separator +
UPLOAD_DIRECTORY;
        File uploadDir = new File(uploadPath);
        if (!uploadDir.exists()) {
            uploadDir.mkdir();
        }

        // Write the file to the specified location
        filePart.write(uploadPath + File.separator + fileName);

        // Respond to client
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
```

```
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head><title>File Upload Result</title></head>");
            out.println("<body>");
            out.println("<h1>File Upload Successful</h1>");
            out.println("<p>File <strong>" + fileName + "</strong> has been uploaded.
</p>");
            if (description != null && !description.trim().isEmpty()) {
                out.println("<p>Description: " + description + "</p>");
            }
            out.println("</body>");
            out.println("</html>");
        }

        // Helper method to extract file name from HTTP header
        private String getSubmittedFileName(Part part) {
            String contentDisp = part.getHeader("content-disposition");
            String[] items = contentDisp.split(";");
            for (String item : items) {
                if (item.trim().startsWith("filename")) {
                    return item.substring(item.indexOf('=') + 2, item.length() - 1);
                }
            }
            return "";
        }
    }
```

Best Practices for Request and Response Handling

1. **Always validate input**: Never trust user input; validate all parameters.
2. **Use appropriate character encoding**: Set response character encoding to UTF-8 for international character support.
3. **Close resources properly**: Always close streams and readers in a finally block or try-with-resources.
4. **Set appropriate content type**: Always set the correct content type for your response.
5. **Use proper HTTP status codes**: Return appropriate status codes for different situations.
6. **Avoid XSS vulnerabilities**: Escape HTML output to prevent cross-site scripting attacks.
7. **Use PRG pattern**: For form submissions, use Post-Redirect-Get pattern to prevent duplicate submissions.
8. **Buffer management**: For large responses, be mindful of buffer size and flushing.
9. **Error handling**: Provide meaningful error messages while not exposing sensitive information.
10. **Secure sensitive data**: Never include sensitive data like passwords in URLs or error messages.

2.4 ServletConfig and ServletContext

ServletConfig

`ServletConfig` is an interface that provides servlet configuration information to a servlet during initialization. Each servlet has its own ServletConfig object.

Key Methods:

```
String getServletName();                    // Returns the servlet's name
ServletContext getServletContext();         // Returns the ServletContext
String getInitParameter(String name);       // Returns an init parameter value
Enumeration<String> getInitParameterNames(); // Returns all init parameter names
```

How to Use ServletConfig:

```
@WebServlet(
    name = "ConfigDemoServlet",
    urlPatterns = {"/config-demo"},
    initParams = {
        @WebInitParam(name = "database", value =
"jdbc:mysql://localhost:3306/mydb"),
        @WebInitParam(name = "username", value = "admin")
    }
)
public class ConfigDemoServlet extends HttpServlet {

    private String database;
    private String username;

    @Override
    public void init() throws ServletException {
        // Get initialization parameters from ServletConfig
        ServletConfig config = getServletConfig();
        this.database = config.getInitParameter("database");
        this.username = config.getInitParameter("username");

        // Log initialization
        System.out.println("ConfigDemoServlet initialized with database: " +
database);
    }

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

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>ServletConfig Demo</title></head>");
        out.println("<body>");
        out.println("<h1>ServletConfig Demonstration</h1>");

        out.println("<h2>Servlet Information</h2>");
        out.println("<ul>");
```

```
        out.println("<li>Servlet Name: " + getServletConfig().getServletName() + "
</li>");
        out.println("<li>Database: " + database + "</li>");
        out.println("<li>Username: " + username + "</li>");
        out.println("</ul>");

        out.println("<h2>All Init Parameters</h2>");
        out.println("<ul>");
        Enumeration<String> paramNames =
getServletConfig().getInitParameterNames();
        while (paramNames.hasMoreElements()) {
            String paramName = paramNames.nextElement();
            String paramValue = getServletConfig().getInitParameter(paramName);
            out.println("<li>" + paramName + ": " + paramValue + "</li>");
        }
        out.println("</ul>");

        out.println("</body>");
        out.println("</html>");
    }
}
```

Configuring ServletConfig in web.xml:

```
<servlet>
    <servlet-name>ConfigDemoServlet</servlet-name>
    <servlet-class>com.example.ConfigDemoServlet</servlet-class>
    <init-param>
        <param-name>database</param-name>
        <param-value>jdbc:mysql://localhost:3306/mydb</param-value>
    </init-param>
    <init-param>
        <param-name>username</param-name>
        <param-value>admin</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>ConfigDemoServlet</servlet-name>
    <url-pattern>/config-demo</url-pattern>
</servlet-mapping>
```

ServletContext

`ServletContext` represents the entire web application and is shared by all servlets in the application. It provides methods to access application-level information and resources.

Key Methods:

```
String getContextPath();                        // Returns the context path
String getServerInfo();                         // Returns server info
String getMimeType(String file);                // Returns MIME type for a file
String getRealPath(String path);                // Returns file system path
URL getResource(String path);                   // Returns a URL to a resource
InputStream getResourceAsStream(String path);   // Returns a resource as a stream
RequestDispatcher getRequestDispatcher(String path);  // Returns a request
dispatcher

// Context initialization parameters
String getInitParameter(String name);           // Returns a context init
parameter
Enumeration<String> getInitParameterNames();    // Returns all context param names

// Context attributes (application-scope)
Object getAttribute(String name);               // Gets an attribute
void setAttribute(String name, Object value);   // Sets an attribute
void removeAttribute(String name);              // Removes an attribute
Enumeration<String> getAttributeNames();        // Returns all attribute names

// Logging
void log(String message);                       // Logs a message
void log(String message, Throwable throwable); // Logs a message with exception

// Servlet API 3.0+ additions
Set<String> getResourcePaths(String path);      // Gets all resources under a path
SessionCookieConfig getSessionCookieConfig();   // Gets session cookie config
void addListener(EventListener listener);       // Adds a context listener
void addFilter(String name, Filter filter);     // Adds a filter
void addServlet(String name, Servlet servlet); // Adds a servlet
```

How to Use ServletContext:

```
@WebServlet("/context-demo")
public class ContextDemoServlet extends HttpServlet {

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

        // Get the ServletContext
        ServletContext context = getServletContext();

        // Example of accessing context parameters
        String dbUrl = context.getInitParameter("dbUrl");

        // Example of using context attributes
        Integer visitorCount = (Integer) context.getAttribute("visitorCount");
        if (visitorCount == null) {
            visitorCount = 1;
        } else {
```

```
            visitorCount++;
        }
        context.setAttribute("visitorCount", visitorCount);

        // Example of using resource methods
        String configPath = context.getRealPath("/WEB-INF/config.properties");
        Set<String> resourcePaths = context.getResourcePaths("/WEB-INF/");

        // Log an application message
        context.log("ContextDemoServlet accessed, visitor count: " +
visitorCount);

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>ServletContext Demo</title></head>");
        out.println("<body>");
        out.println("<h1>ServletContext Demonstration</h1>");

        out.println("<h2>Application Information</h2>");
        out.println("<ul>");
        out.println("<li>Context Path: " + context.getContextPath() + "</li>");
        out.println("<li>Server Info: " + context.getServerInfo() + "</li>");
        out.println("<li>Servlet API Version: "
                + context.getMajorVersion() + "." + context.getMinorVersion() + "
</li>");
        out.println("<li>Database URL: " + dbUrl + "</li>");
        out.println("<li>Visitor Count: " + visitorCount + "</li>");
        out.println("</ul>");

        out.println("<h2>Resources in /WEB-INF/</h2>");
        out.println("<ul>");
        if (resourcePaths != null) {
            for (String path : resourcePaths) {
                out.println("<li>" + path + "</li>");
            }
        }
        out.println("</ul>");

        out.println("<h2>Config File Path</h2>");
        out.println("<p>" + configPath + "</p>");

        out.println("</body>");
        out.println("</html>");
    }
}
```

Configuring ServletContext Parameters in web.xml:

```
<context-param>
    <param-name>dbUrl</param-name>
    <param-value>jdbc:mysql://localhost:3306/myapp</param-value>
</context-param>

<context-param>
    <param-name>adminEmail</param-name>
    <param-value>admin@example.com</param-value>
</context-param>
```

Key Differences Between ServletConfig and ServletContext

Feature	ServletConfig	ServletContext
Scope	Servlet-specific	Application-wide
Instance	One per servlet	One per web application
Lifecycle	Created during servlet initialization	Created when application starts
Access	Only accessible within the servlet	Accessible from any servlet/JSP
Usage	Configuration specific to a servlet	Shared application configuration
Methods	Limited to initialization parameters	Rich set of application methods

Best Practices

1. **Use ServletConfig for**:
    - Servlet-specific configuration
    - Information needed only by a single servlet
    - Initialization data that varies between servlet instances
2. **Use ServletContext for**:
    - Application-wide configuration
    - Sharing data between servlets
    - Accessing application resources
    - Application-level logging
    - Counting application statistics
3. **Avoid Storing Too Much in ServletContext**:
    - Store only application-level data
    - Don't use it as a cache for large objects
    - Consider thread safety when storing mutable objects
4. **Thread Safety**:
    - ServletContext is shared by all servlets, making thread safety critical

- Use synchronization when updating shared objects in context
- Consider using concurrent collections for shared data

5. **Parameter Usage**:
   - Use meaningful parameter names
   - Document all parameters
   - Provide sensible defaults if parameters are missing

2.5 Session Management Techniques

Session management is essential for tracking user interactions across multiple requests. Java servlets provide several methods for maintaining session information.

1. HTTP Cookies

Cookies are small pieces of data sent from the server and stored on the client's browser.

Creating and Using Cookies:

```
@WebServlet("/cookie-demo")
public class CookieDemoServlet extends HttpServlet {

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

        // Check if user already has a visit cookie
        Cookie[] cookies = request.getCookies();
        boolean returning = false;
        int visitCount = 1;

        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if ("visitCount".equals(cookie.getName())) {
                    returning = true;
                    visitCount = Integer.parseInt(cookie.getValue()) + 1;
                    break;
                }
            }
        }

        // Set or update the visit cookie
        Cookie visitCookie = new Cookie("visitCount", String.valueOf(visitCount));
        visitCookie.setMaxAge(60 * 60 * 24 * 30); // 30 days
        visitCookie.setPath("/");
        response.addCookie(visitCookie);

        // Set preferences cookie (example of additional cookie)
        String theme = request.getParameter("theme");
        if (theme != null && (theme.equals("light") || theme.equals("dark"))) {
            Cookie themeCookie = new Cookie("theme", theme);
```

```
            themeCookie.setMaxAge(60 * 60 * 24 * 365); // 1 year
            themeCookie.setPath("/");
            response.addCookie(themeCookie);
        }

        // Get current theme from cookies
        String currentTheme = "light"; // default
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if ("theme".equals(cookie.getName())) {
                    currentTheme = cookie.getValue();
                    break;
                }
            }
        }

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Cookie Demo</title>");
        out.println("<style>");
        if ("dark".equals(currentTheme)) {
            out.println("body { background-color: #333; color: #fff; }");
        } else {
            out.println("body { background-color: #fff; color: #333; }");
        }
        out.println("</style>");
        out.println("</head>");
        out.println("<body>");

        if (returning) {
            out.println("<h1>Welcome Back!</h1>");
            out.println("<p>You have visited this page " + visitCount + " times.
</p>");
        } else {
            out.println("<h1>Welcome to Our Site!</h1>");
            out.println("<p>This is your first visit.</p>");
        }

        out.println("<h2>Change Theme</h2>");
        out.println("<a href='?theme=light'>Light Theme</a> | ");
        out.println("<a href='?theme=dark'>Dark Theme</a>");

        out.println("<h2>All Cookies</h2>");
        out.println("<ul>");
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                out.println("<li>" + cookie.getName() + ": " + cookie.getValue() +
"</li>");
            }
```

```
            } else {
                out.println("<li>No cookies found</li>");
            }
            out.println("</ul>");

            out.println("</body>");
            out.println("</html>");
        }
    }
```

Cookie Attributes:

```
Cookie cookie = new Cookie("name", "value");
cookie.setDomain(".example.com");            // Domain for which cookie is valid
cookie.setPath("/");                         // Path for which cookie is valid
cookie.setMaxAge(60 * 60 * 24);              // Cookie lifespan in seconds (1 day
here)
cookie.setSecure(true);                      // Send only via HTTPS
cookie.setHttpOnly(true);                    // Not accessible via JavaScript
(prevent XSS)
cookie.setComment("User tracking cookie");   // Description of the cookie
```

Cookie Limitations:

- Limited size (usually 4KB)
- Clients may disable or clear cookies
- Not secure for sensitive data unless properly configured
- Limited to around 50 cookies per domain

2. HttpSession

HttpSession provides a way to track a user across multiple requests, storing session data on the server.

Using HttpSession:

```
@WebServlet("/session-demo")
public class SessionDemoServlet extends HttpServlet {

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

        // Get or create session
        HttpSession session = request.getSession();

        // Check if session is new
        boolean isNewSession = session.isNew();

        // Get session information
```

```
        String sessionId = session.getId();
        long creationTime = session.getCreationTime();
        long lastAccessedTime = session.getLastAccessedTime();

        // Set or get attributes in session
        Integer visitCount = (Integer) session.getAttribute("visitCount");
        if (visitCount == null) {
            visitCount = 1;
        } else {
            visitCount++;
        }
        session.setAttribute("visitCount", visitCount);

        // Get user preference from session or request
        String username = (String) session.getAttribute("username");
        String color = (String) session.getAttribute("color");

        // Update preferences from request parameters
        String newUsername = request.getParameter("username");
        if (newUsername != null && !newUsername.trim().isEmpty()) {
            username = newUsername;
            session.setAttribute("username", username);
        }

        String newColor = request.getParameter("color");
        if (newColor != null && (newColor.equals("red") || newColor.equals("blue")
|| newColor.equals("green"))) {
            color = newColor;
            session.setAttribute("color", color);
        }

        // Handle logout
        String action = request.getParameter("action");
        if ("logout".equals(action)) {
            session.invalidate();
            response.sendRedirect(request.getContextPath() + "/session-demo");
            return;
        }

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Session Demo</title>");
        out.println("<style>");
        if (color != null) {
            out.println("body { color: " + color + "; }");
        }
        out.println("</style>");
        out.println("</head>");
        out.println("<body>");
```

```
        if (isNewSession) {
            out.println("<h1>Welcome to Session Demo</h1>");
        } else {
            out.println("<h1>Welcome Back to Session Demo</h1>");
            if (username != null) {
                out.println("<p>Hello, " + username + "!</p>");
            }
        }

        out.println("<h2>Session Information</h2>");
        out.println("<ul>");
        out.println("<li>Session ID: " + sessionId + "</li>");
        out.println("<li>Created: " + new Date(creationTime) + "</li>");
        out.println("<li>Last Accessed: " + new Date(lastAccessedTime) + "</li>");
        out.println("<li>Visit Count: " + visitCount + "</li>");
        out.println("</ul>");

        out.println("<h2>Preferences</h2>");
        out.println("<form action='session-demo' method='get'>");
        out.println("  <div>");
        out.println("    <label for='username'>Username:</label>");
        out.println("    <input type='text' id='username' name='username' value='"
                + (username != null ? username : "") + "'>");
        out.println("  </div>");
        out.println("  <div>");
        out.println("    <label>Text Color:</label>");
        out.println("    <input type='radio' id='red' name='color' value='red'"
                + ("red".equals(color) ? " checked" : "") + ">");
        out.println("    <label for='red'>Red</label>");
        out.println("    <input type='radio' id='blue' name='color' value='blue'"
                + ("blue".equals(color) ? " checked" : "") + ">");
        out.println("    <label for='blue'>Blue</label>");
        out.println("    <input type='radio' id='green' name='color'
value='green'"
                + ("green".equals(color) ? " checked" : "") + ">");
        out.println("    <label for='green'>Green</label>");
        out.println("  </div>");
        out.println("  <div>");
        out.println("    <input type='submit' value='Save Preferences'>");
        out.println("  </div>");
        out.println("</form>");

        out.println("<p><a href='?action=logout'>Logout</a></p>");

        out.println("</body>");
        out.println("</html>");
    }
}
```

HttpSession Methods:

```
HttpSession session = request.getSession();      // Get or create session
HttpSession session = request.getSession(true);  // Get or create session
HttpSession session = request.getSession(false); // Get session, null if doesn't
exist

String id = session.getId();                     // Get session ID
boolean isNew = session.isNew();                 // Check if session is new
long creationTime = session.getCreationTime();   // Get creation time
long lastAccess = session.getLastAccessedTime(); // Get last accessed time

session.setAttribute("name", value);             // Store data in session
Object value = session.getAttribute("name");     // Get data from session
Enumeration<String> names = session.getAttributeNames();  // Get all attribute
names
session.removeAttribute("name");                 // Remove data from session

session.setMaxInactiveInterval(30 * 60);         // Set timeout (seconds)
int timeout = session.getMaxInactiveInterval();  // Get timeout

session.invalidate();                            // Invalidate session
```

Session Configuration in web.xml:

```
<session-config>
    <!-- Session timeout in minutes -->
    <session-timeout>30</session-timeout>

    <!-- Configure session tracking methods (Servlet 3.0+) -->
    <tracking-mode>COOKIE</tracking-mode>
    <tracking-mode>URL</tracking-mode>

    <!-- Configure session cookies (Servlet 3.0+) -->
    <cookie-config>
        <name>JSESSIONID</name>
        <domain>example.com</domain>
        <path>/</path>
        <http-only>true</http-only>
        <secure>true</secure>
        <max-age>86400</max-age>
    </cookie-config>
</session-config>
```

3. URL Rewriting

URL rewriting appends session information to URLs when cookies are disabled.

Using URL Rewriting:

```
@WebServlet("/url-demo")
public class UrlRewritingServlet extends HttpServlet {

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

        // Get or create session
        HttpSession session = request.getSession();

        // Store sample data in session
        Integer counter = (Integer) session.getAttribute("counter");
        if (counter == null) {
            counter = 1;
        } else {
            counter++;
        }
        session.setAttribute("counter", counter);

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>URL Rewriting Demo</title></head>");
        out.println("<body>");

        out.println("<h1>URL Rewriting Demonstration</h1>");
        out.println("<p>Counter: " + counter + "</p>");

        // Create URL-rewritten links (encodeURL handles the rewriting logic)
        String incrementUrl = response.encodeURL(request.getContextPath() + "/url-
demo?action=increment");
        String resetUrl = response.encodeURL(request.getContextPath() + "/url-
demo?action=reset");

        out.println("<p>");
        out.println("<a href='" + incrementUrl + "'>Increment Counter</a> | ");
        out.println("<a href='" + resetUrl + "'>Reset Counter</a>");
        out.println("</p>");

        // Show session ID
        out.println("<p>Session ID: " + session.getId() + "</p>");
        out.println("<p>Note: If cookies are disabled, the session ID will be
appended to the URLs.</p>");

        out.println("</body>");
        out.println("</html>");

        // Handle actions
        String action = request.getParameter("action");
        if ("increment".equals(action)) {
```

```
            session.setAttribute("counter", counter + 1);
        } else if ("reset".equals(action)) {
            session.setAttribute("counter", 0);
        }
    }
}
```

Key Methods for URL Rewriting:

```
// For URLs
String encodedURL = response.encodeURL(originalURL);

// For redirects
String encodedRedirectURL = response.encodeRedirectURL(redirectURL);
```

4. Hidden Form Fields

Hidden form fields can be used to maintain state between requests.

Using Hidden Form Fields:

```
@WebServlet("/hidden-form-demo")
public class HiddenFormFieldsServlet extends HttpServlet {

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

        // Get parameters from request (or set defaults)
        String username = request.getParameter("username");
        String step = request.getParameter("step");

        if (username == null) {
            username = "";
        }

        if (step == null) {
            step = "1";
        }

        // Respond based on step
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>Hidden Form Fields Demo</title></head>");
        out.println("<body>");
```

```
        out.println("<h1>Multi-step Form with Hidden Fields</h1>");

        if ("1".equals(step)) {
            // Step 1: Collect username
            out.println("<h2>Step 1: Enter Your Username</h2>");
            out.println("<form action='hidden-form-demo' method='post'>");
            out.println("  <div>");
            out.println("    <label for='username'>Username:</label>");
            out.println("    <input type='text' id='username' name='username'
required>");
            out.println("  </div>");
            out.println("  <input type='hidden' name='step' value='2'>");
            out.println("  <div>");
            out.println("    <input type='submit' value='Next'>");
            out.println("  </div>");
            out.println("</form>");

        } else if ("2".equals(step)) {
            // Step 2: Collect preferences
            out.println("<h2>Step 2: Select Preferences</h2>");
            out.println("<form action='hidden-form-demo' method='post'>");
            out.println("  <div>");
            out.println("    <label>Favorite Color:</label>");
            out.println("    <select name='color'>");
            out.println("      <option value='red'>Red</option>");
            out.println("      <option value='blue'>Blue</option>");
            out.println("      <option value='green'>Green</option>");
            out.println("    </select>");
            out.println("  </div>");
            out.println("  <div>");
            out.println("    <label>Notification Preferences:</label><br>");
            out.println("    <input type='checkbox' id='email'
name='notifications' value='email'>");
            out.println("    <label for='email'>Email</label><br>");
            out.println("    <input type='checkbox' id='sms' name='notifications'
value='sms'>");
            out.println("    <label for='sms'>SMS</label>");
            out.println("  </div>");
            out.println("  <input type='hidden' name='username' value='" +
username + "'>");
            out.println("  <input type='hidden' name='step' value='3'>");
            out.println("  <div>");
            out.println("    <input type='submit' value='Next'>");
            out.println("  </div>");
            out.println("</form>");

        } else if ("3".equals(step)) {
            // Step 3: Confirmation
            String color = request.getParameter("color");
            String[] notifications = request.getParameterValues("notifications");

            out.println("<h2>Step 3: Confirmation</h2>");
            out.println("<p>Thank you for your submission. Here's what we've
received:</p>");
```

```
            out.println("<ul>");
            out.println("  <li><strong>Username:</strong> " + username + "</li>");

            if (color != null) {
                out.println("  <li><strong>Favorite Color:</strong> " + color + "
</li>");
            }

            out.println("  <li><strong>Notifications:</strong> ");
            if (notifications != null && notifications.length > 0) {
                out.println(String.join(", ", notifications));
            } else {
                out.println("None selected");
            }
            out.println("  </li>");
            out.println("</ul>");

            out.println("<p><a href='hidden-form-demo'>Start Over</a></p>");
        }

        out.println("</body>");
        out.println("</html>");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        // Forward POST requests to doGet for simplicity
        doGet(request, response);
    }
}
```

5. Session Management Best Practices

1. **Security Considerations**:
    - Use HTTPS for secure session transmission
    - Set HttpOnly flag to prevent JavaScript access to cookies
    - Set Secure flag to ensure cookies are sent only over HTTPS
    - Regenerate session IDs after login to prevent session fixation
    - Implement proper timeout and session expiration
2. **Performance Considerations**:
    - Store only necessary data in session
    - Use appropriate session timeout values
    - Consider using distributed session management for clustered environments
    - Implement session cleanup strategies
3. **Reliability**:

- Implement fallback mechanisms when cookies are disabled
- Validate session data before use
- Handle session timeout gracefully

4. **Implementation Guidelines**:

- Use HttpSession for complex state management
- Use cookies for simple persistent preferences
- Use hidden fields for multi-step forms
- Consider using a combination of techniques for robustness

6. Session Data Storage Options

1. **In-Memory Storage**:

- Default for most containers
- Fast but limited by server memory
- Lost on server restart
- Not suitable for clustering without additional configuration

2. **Database Storage**:

- Persistent across server restarts
- Suitable for clustering
- Requires custom implementation or container support
- Slower than in-memory storage

3. **Distributed Caches**:

- Options like Redis, Memcached, Hazelcast
- Good for clustered environments
- Fast and scalable
- Requires additional configuration

Example of Custom Session Management with Database:

```
@WebListener
public class DatabaseSessionManager implements HttpSessionListener,
HttpSessionAttributeListener {

    private DataSource dataSource; // Initialized in constructor or from JNDI

    @Override
    public void sessionCreated(HttpSessionEvent se) {
        HttpSession session = se.getSession();
        try (Connection conn = dataSource.getConnection();
             PreparedStatement ps = conn.prepareStatement(
                 "INSERT INTO sessions (session_id, creation_time) VALUES (?,
?)")) {
            ps.setString(1, session.getId());
            ps.setLong(2, session.getCreationTime());
```

```
            ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        HttpSession session = se.getSession();
        try (Connection conn = dataSource.getConnection();
             PreparedStatement ps = conn.prepareStatement(
                 "DELETE FROM sessions WHERE session_id = ?")) {
            ps.setString(1, session.getId());
            ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void attributeAdded(HttpSessionBindingEvent event) {
        saveAttribute(event.getSession().getId(), event.getName(),
event.getValue());
    }

    @Override
    public void attributeReplaced(HttpSessionBindingEvent event) {
        saveAttribute(event.getSession().getId(), event.getName(),
event.getValue());
    }

    @Override
    public void attributeRemoved(HttpSessionBindingEvent event) {
        removeAttribute(event.getSession().getId(), event.getName());
    }

    private void saveAttribute(String sessionId, String name, Object value) {
        try (Connection conn = dataSource.getConnection();
             PreparedStatement ps = conn.prepareStatement(
                 "REPLACE INTO session_attributes (session_id, attr_name,
attr_value) VALUES (?, ?, ?)")) {
            ps.setString(1, sessionId);
            ps.setString(2, name);
            ps.setBytes(3, serializeObject(value));
            ps.executeUpdate();
        } catch (SQLException | IOException e) {
            e.printStackTrace();
        }
    }

    private void removeAttribute(String sessionId, String name) {
        try (Connection conn = dataSource.getConnection();
             PreparedStatement ps = conn.prepareStatement(
                 "DELETE FROM session_attributes WHERE session_id = ? AND
```

```
            attr_name = ?")) {
                ps.setString(1, sessionId);
                ps.setString(2, name);
                ps.executeUpdate();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        // Helper method to serialize object
        private byte[] serializeObject(Object obj) throws IOException {
            try (ByteArrayOutputStream baos = new ByteArrayOutputStream();
                 ObjectOutputStream oos = new ObjectOutputStream(baos)) {
                oos.writeObject(obj);
                return baos.toByteArray();
            }
        }
    }
```

2.6 Filters and Listeners

Filters and listeners provide powerful mechanisms for intercepting and monitoring servlet container events.

Servlet Filters

Filters intercept requests to and responses from servlets, allowing preprocessing and postprocessing of request and response objects.

Filter Lifecycle:

1. Init - Called when the filter is initialized
2. doFilter - Called for each request that matches the filter mapping
3. Destroy - Called when the filter is being taken out of service

Creating a Basic Filter:

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

@WebFilter("/*")
public class LoggingFilter implements Filter {

    private FilterConfig filterConfig;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
        filterConfig.getServletContext().log("LoggingFilter initialized");
    }
```

```
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        String requestURI = httpRequest.getRequestURI();

        // Log before processing
        filterConfig.getServletContext().log("Request received for: " +
requestURI);
        long startTime = System.currentTimeMillis();

        // Pass request to next filter or servlet
        chain.doFilter(request, response);

        // Log after processing
        long endTime = System.currentTimeMillis();
        filterConfig.getServletContext().log("Request completed for: " +
requestURI +
                ", processing time: " + (endTime - startTime) + "ms");
    }

    @Override
    public void destroy() {
        filterConfig.getServletContext().log("LoggingFilter destroyed");
        this.filterConfig = null;
    }
}
```

Common Filter Use Cases:

1. **Authentication and Authorization**:

```
@WebFilter("/*")
public class AuthenticationFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Get the requested URI
        String requestURI = httpRequest.getRequestURI();

        // Skip filter for public resources and login page
        if (isPublicResource(requestURI)) {
            chain.doFilter(request, response);
```

```
            return;
        }

        // Check if user is authenticated
        HttpSession session = httpRequest.getSession(false);
        boolean isLoggedIn = session != null && session.getAttribute("user") !=
null;

        if (isLoggedIn) {
            // User is authenticated, proceed with the request
            chain.doFilter(request, response);
        } else {
            // User is not authenticated, redirect to login page
            httpResponse.sendRedirect(httpRequest.getContextPath() +
"/login.jsp");
        }
    }

    private boolean isPublicResource(String uri) {
        return uri.endsWith("login.jsp") ||
               uri.endsWith("register.jsp") ||
               uri.endsWith(".css") ||
               uri.endsWith(".js") ||
               uri.endsWith(".png") ||
               uri.endsWith(".jpg") ||
               uri.contains("/public/");
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code if needed
    }

    @Override
    public void destroy() {
        // Cleanup code if needed
    }
}
```

2. **Request/Response Modification**:

```
@WebFilter("/*")
public class EncodingFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        // Set character encoding for request
        if (request.getCharacterEncoding() == null) {
            request.setCharacterEncoding("UTF-8");
```

```
        }

        // Set character encoding and content type for response
        response.setCharacterEncoding("UTF-8");

        // Continue with the filter chain
        chain.doFilter(request, response);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

3. **Response Compression**:

```
@WebFilter("/*")
public class CompressionFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Check if browser supports GZIP
        String acceptEncoding = httpRequest.getHeader("Accept-Encoding");
        boolean supportsGzip = acceptEncoding != null &&
acceptEncoding.contains("gzip");

        if (supportsGzip) {
            // Create a wrapped response that compresses the output
            GZipServletResponseWrapper gzipResponse = new
GZipServletResponseWrapper(httpResponse);

            // Set content encoding header
            gzipResponse.setHeader("Content-Encoding", "gzip");

            // Process the request/response with the filter chain
            chain.doFilter(request, gzipResponse);

            // Finish the response
            gzipResponse.finish();
        } else {
```

```
            // Continue without compression if GZIP is not supported
            chain.doFilter(request, response);
        }
    }

    // Inner wrapper class that implements GZIP compression
    // Note: This is a simplified implementation. In practice, you'd use a library
or implement the full wrapper
    class GZipServletResponseWrapper extends HttpServletResponseWrapper {
        private GZIPOutputStream gzipOutputStream;
        private ServletOutputStream servletOutputStream;
        private PrintWriter printWriter;

        public GZipServletResponseWrapper(HttpServletResponse response) {
            super(response);
        }

        @Override
        public ServletOutputStream getOutputStream() throws IOException {
            if (printWriter != null) {
                throw new IllegalStateException("getWriter() has already been
called");
            }

            if (servletOutputStream == null) {
                servletOutputStream = getResponse().getOutputStream();
                gzipOutputStream = new GZIPOutputStream(servletOutputStream);
            }

            return new ServletOutputStreamWrapper(gzipOutputStream);
        }

        @Override
        public PrintWriter getWriter() throws IOException {
            if (servletOutputStream != null) {
                throw new IllegalStateException("getOutputStream() has already
been called");
            }

            if (printWriter == null) {
                servletOutputStream = getResponse().getOutputStream();
                gzipOutputStream = new GZIPOutputStream(servletOutputStream);
                printWriter = new PrintWriter(new
OutputStreamWriter(gzipOutputStream, "UTF-8"));
            }

            return printWriter;
        }

        public void finish() throws IOException {
            if (printWriter != null) {
                printWriter.close();
            } else if (gzipOutputStream != null) {
                gzipOutputStream.close();
```

```
            }
        }

        // Implementing a wrapper for ServletOutputStream
        class ServletOutputStreamWrapper extends ServletOutputStream {
            private final OutputStream outputStream;

            public ServletOutputStreamWrapper(OutputStream outputStream) {
                this.outputStream = outputStream;
            }

            @Override
            public void write(int b) throws IOException {
                outputStream.write(b);
            }

            @Override
            public boolean isReady() {
                return true;
            }

            @Override
            public void setWriteListener(WriteListener writeListener) {
                throw new UnsupportedOperationException("Not implemented");
            }
        }
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

4. **Caching Filter**:

```
@WebFilter("/*")
public class CacheControlFilter implements Filter {

    private long defaultMaxAge = 3600; // 1 hour in seconds

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
```

```
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        String uri = httpRequest.getRequestURI();

        // Apply caching for static resources
        if (isStaticResource(uri)) {
            // Set cache control headers
            httpResponse.setHeader("Cache-Control", "public, max-age=" +
defaultMaxAge);
            // Set expiration date
            httpResponse.setDateHeader("Expires", System.currentTimeMillis() +
defaultMaxAge * 1000);
        } else {
            // Prevent caching for dynamic content
            httpResponse.setHeader("Cache-Control", "no-store, no-cache, must-
revalidate, max-age=0");
            httpResponse.setHeader("Pragma", "no-cache");
            httpResponse.setDateHeader("Expires", 0);
        }

        // Continue with the filter chain
        chain.doFilter(request, response);
    }

    private boolean isStaticResource(String uri) {
        return uri.endsWith(".css") ||
               uri.endsWith(".js") ||
               uri.endsWith(".png") ||
               uri.endsWith(".jpg") ||
               uri.endsWith(".gif") ||
               uri.endsWith(".ico");
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        String maxAgeParam = filterConfig.getInitParameter("maxAge");
        if (maxAgeParam != null) {
            try {
                defaultMaxAge = Long.parseLong(maxAgeParam);
            } catch (NumberFormatException e) {
                // Use default if parameter is invalid
            }
        }
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

Filter Mapping in web.xml:

```
<filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>com.example.filters.AuthenticationFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>
    <url-pattern>/admin/*</url-pattern>
    <url-pattern>/secure/*</url-pattern>
</filter-mapping>

<filter>
    <filter-name>EncodingFilter</filter-name>
    <filter-class>com.example.filters.EncodingFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>EncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter>
    <filter-name>CompressionFilter</filter-name>
    <filter-class>com.example.filters.CompressionFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>CompressionFilter</filter-name>
    <url-pattern>*.html</url-pattern>
    <url-pattern>*.css</url-pattern>
    <url-pattern>*.js</url-pattern>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

Filter Ordering:

The order in which filters are applied is determined by:

1. For web.xml configuration: The order of `<filter-mapping>` elements
2. For annotation-based configuration: Using `@WebFilter(filterName="name", urlPatterns={"/*"})` with a deployment descriptor fragment that specifies ordering via `<absolute-ordering>` or `<relative-ordering>`

Filter Order Example in web.xml:

```
<absolute-ordering>
    <name>SecurityFilter</name>
    <name>LoggingFilter</name>
    <name>EncodingFilter</name>
</absolute-ordering>
```

Servlet Listeners

Listeners allow you to monitor and react to events in the servlet container lifecycle.

Types of Servlet Listeners:

1. **ServletContextListener**: Notified of ServletContext lifecycle changes (application startup/shutdown)
2. **ServletContextAttributeListener**: Notified of ServletContext attribute changes
3. **HttpSessionListener**: Notified of HttpSession creation/destruction
4. **HttpSessionAttributeListener**: Notified of HttpSession attribute changes
5. **HttpSessionBindingListener**: Notified when an object is bound/unbound from a session
6. **HttpSessionActivationListener**: Notified when a session is activated/passivated
7. **ServletRequestListener**: Notified of ServletRequest creation/destruction
8. **ServletRequestAttributeListener**: Notified of ServletRequest attribute changes

Creating Servlet Listeners:

1. **ServletContextListener Example**:

```
import javax.servlet.*;
import javax.servlet.annotation.*;

@WebListener
public class AppInitializer implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        // Called when the web application is starting up
        ServletContext context = sce.getServletContext();

        // Initialize application-wide resources
        context.log("Application starting up");

        // Example: Initialize a database connection pool
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            context.log("JDBC Driver loaded successfully");

            // Store configuration in context
            context.setAttribute("appStartTime", System.currentTimeMillis());
            context.setAttribute("appName", "MyWebApp");
        } catch (ClassNotFoundException e) {
            context.log("Error loading JDBC driver", e);
        }
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        // Called when the web application is shutting down
        ServletContext context = sce.getServletContext();

        // Clean up application resources
        context.log("Application shutting down");
```

```
        // Example: Close database connections, release resources
        Long startTime = (Long) context.getAttribute("appStartTime");
        if (startTime != null) {
            long uptime = System.currentTimeMillis() - startTime;
            context.log("Application uptime: " + (uptime / 1000) + " seconds");
        }
    }
}
```

2. **HttpSessionListener Example**:

```
import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebListener
public class SessionManager implements HttpSessionListener {

    private static int activeSessions = 0;

    @Override
    public void sessionCreated(HttpSessionEvent se) {
        // Called when a new session is created
        synchronized (this) {
            activeSessions++;
        }

        HttpSession session = se.getSession();
        session.setMaxInactiveInterval(30 * 60); // 30 minutes

        // Log session creation
        session.getServletContext().log("Session created: " + session.getId() +
                ", Active sessions: " + activeSessions);
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        // Called when a session is invalidated or times out
        synchronized (this) {
            activeSessions--;
            if (activeSessions < 0) activeSessions = 0;
        }

        HttpSession session = se.getSession();

        // Log session destruction
        session.getServletContext().log("Session destroyed: " + session.getId() +
                ", Active sessions: " + activeSessions);
    }

    public static int getActiveSessions() {
        return activeSessions;
```

```
    }
}
```

3. **ServletRequestListener Example**:

```
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebListener
public class RequestMonitor implements ServletRequestListener {

    @Override
    public void requestInitialized(ServletRequestEvent sre) {
        // Called when a request is initialized
        ServletRequest request = sre.getServletRequest();

        // For HTTP requests
        if (request instanceof HttpServletRequest) {
            HttpServletRequest httpRequest = (HttpServletRequest) request;

            // Store request start time
            request.setAttribute("requestStartTime", System.currentTimeMillis());

            // Log request information
            String uri = httpRequest.getRequestURI();
            String method = httpRequest.getMethod();
            String remoteAddr = httpRequest.getRemoteAddr();

            ServletContext context = sre.getServletContext();
            context.log("Request initialized: " + method + " " + uri + " from " +
remoteAddr);
        }
    }

    @Override
    public void requestDestroyed(ServletRequestEvent sre) {
        // Called when a request is destroyed
        ServletRequest request = sre.getServletRequest();
        Long startTime = (Long) request.getAttribute("requestStartTime");

        if (startTime != null) {
            long duration = System.currentTimeMillis() - startTime;

            // For HTTP requests
            if (request instanceof HttpServletRequest) {
                HttpServletRequest httpRequest = (HttpServletRequest) request;
                String uri = httpRequest.getRequestURI();

                ServletContext context = sre.getServletContext();
                context.log("Request completed: " + uri + " in " + duration +
"ms");
```

```
            }
        }
    }
}
```

4. **HttpSessionAttributeListener Example**:

```
import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebListener
public class SessionAttributeMonitor implements HttpSessionAttributeListener {

    @Override
    public void attributeAdded(HttpSessionBindingEvent event) {
        // Called when an attribute is added to a session
        String name = event.getName();
        Object value = event.getValue();

        HttpSession session = event.getSession();
        session.getServletContext().log("Session attribute added: " + name +
                ", Value type: " + value.getClass().getName() +
                ", Session ID: " + session.getId());
    }

    @Override
    public void attributeRemoved(HttpSessionBindingEvent event) {
        // Called when an attribute is removed from a session
        String name = event.getName();

        HttpSession session = event.getSession();
        session.getServletContext().log("Session attribute removed: " + name +
                ", Session ID: " + session.getId());
    }

    @Override
    public void attributeReplaced(HttpSessionBindingEvent event) {
        // Called when an attribute is replaced in a session
        String name = event.getName();
        Object value = event.getValue();

        HttpSession session = event.getSession();
        session.getServletContext().log("Session attribute replaced: " + name +
                ", New value type: " + value.getClass().getName() +
                ", Session ID: " + session.getId());
    }
}
```

Listener Configuration in web.xml:

```
<listener>
    <listener-class>com.example.listeners.AppInitializer</listener-class>
</listener>

<listener>
    <listener-class>com.example.listeners.SessionManager</listener-class>
</listener>

<listener>
    <listener-class>com.example.listeners.RequestMonitor</listener-class>
</listener>

<listener>
    <listener-class>com.example.listeners.SessionAttributeMonitor</listener-class>
</listener>
```

Best Practices for Filters and Listeners

1. **Filter Best Practices**:
    - Keep filters focused on a single responsibility
    - Order filters appropriately based on dependencies
    - Be mindful of performance impact, especially for filters that apply to all requests
    - Use init parameters to make filters configurable
    - Always call `chain.doFilter()` unless you intentionally want to block the request
    - Handle exceptions properly to avoid breaking the filter chain
    - Consider thread safety in shared variables
2. **Listener Best Practices**:
    - Use listeners for application initialization and cleanup
    - Keep listener methods efficient as they can impact application performance
    - Separate concerns across different listeners
    - Be careful with session listeners in clustered environments
    - For ServletContextListener, initialize resources in contextInitialized and clean up in contextDestroyed
    - Use appropriate listener types for specific needs
3. **Common Filter Patterns**:
    - Authentication and Authorization Filter
    - Logging and Monitoring Filter
    - Compression Filter
    - Character Encoding Filter
    - Cache Control Filter
    - CORS Filter
    - XSS Protection Filter
    - Request Wrapping Filter (for modifying request/response)

4. **Common Listener Patterns**:
    - Application Initialization
    - Session Management
    - Resource Management
    - Application Metrics
    - Context Configuration

2.7 Asynchronous Processing

Asynchronous processing in servlets (introduced in Servlet 3.0) allows long-running operations to be performed without blocking the request-processing thread.

Why Use Asynchronous Processing?

1. **Improved Scalability**: Frees up server threads for other requests
2. **Better Resource Utilization**: Handles more concurrent connections with fewer threads
3. **Long-Running Operations**: Supports operations that take time (e.g., external API calls, complex computations)
4. **Server Push**: Enables Comet, Server-Sent Events (SSE), and similar techniques

How Asynchronous Processing Works

1. The servlet starts asynchronous processing by calling `request.startAsync()`
2. This returns an `AsyncContext` object that represents the asynchronous operation
3. The original request-handling thread completes and returns to the container's thread pool
4. The operation continues in a separate thread
5. When the operation completes, the response is completed via the AsyncContext

Basic Asynchronous Servlet Example

```
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.concurrent.*;

@WebServlet(urlPatterns = "/async-servlet", asyncSupported = true)
public class AsyncServletExample extends HttpServlet {

    private ExecutorService executor = Executors.newFixedThreadPool(10);

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

        // Start async processing
        final AsyncContext asyncContext = request.startAsync();

        // Set timeout (optional)
```

```
        asyncContext.setTimeout(30000); // 30 seconds

        // Add listeners (optional)
        asyncContext.addListener(new AsyncListener() {
            @Override
            public void onComplete(AsyncEvent event) throws IOException {
                System.out.println("Async operation completed");
            }

            @Override
            public void onTimeout(AsyncEvent event) throws IOException {
                System.out.println("Async operation timed out");
                ServletResponse response = event.getAsyncContext().getResponse();
                response.setContentType("text/html");
                response.getWriter().println("Operation timed out");
                event.getAsyncContext().complete();
            }

            @Override
            public void onError(AsyncEvent event) throws IOException {
                System.out.println("Async operation failed: " +
event.getThrowable().getMessage());
                event.getThrowable().printStackTrace();
            }

            @Override
            public void onStartAsync(AsyncEvent event) throws IOException {
                System.out.println("Async operation started");
            }
        });

        // Process the request asynchronously
        executor.execute(() -> {
            try {
                // Simulate a long-running operation
                Thread.sleep(5000); // 5 seconds

                // Get the response from the async context
                ServletResponse asyncResponse = asyncContext.getResponse();
                asyncResponse.setContentType("text/html");

                PrintWriter out = asyncResponse.getWriter();
                out.println("<!DOCTYPE html>");
                out.println("<html>");
                out.println("<head><title>Async Servlet Result</title></head>");
                out.println("<body>");
                out.println("<h1>Async Processing Complete</h1>");
                out.println("<p>The long-running operation completed after 5
seconds.</p>");
                out.println("<p>Time: " + new java.util.Date() + "</p>");
                out.println("</body>");
                out.println("</html>");

                // Complete the async operation
```

```
                asyncContext.complete();

            } catch (Exception e) {
                e.printStackTrace();
            }
        });

        // Original thread completes and returns to the container's thread pool
        System.out.println("Servlet thread completed, but async operation
continues...");
    }

    @Override
    public void destroy() {
        executor.shutdown();
        super.destroy();
    }
}
```

AsyncContext Methods

```
// Starting async processing
AsyncContext asyncContext = request.startAsync();  // Uses original
request/response
AsyncContext asyncContext = request.startAsync(request, response);  // Uses
provided req/resp

// Setting timeout
asyncContext.setTimeout(timeoutMillis);

// Getting request/response
ServletRequest request = asyncContext.getRequest();
ServletResponse response = asyncContext.getResponse();

// Completing the async operation
asyncContext.complete();

// Dispatching to another resource
asyncContext.dispatch();  // Dispatch to the same URL
asyncContext.dispatch(path);  // Dispatch to the specified path

// Adding listeners
asyncContext.addListener(asyncListener);
asyncContext.addListener(asyncListener, request, response);

// Starting async I/O operations
ServletInputStream input = request.getInputStream();
ServletOutputStream output = response.getOutputStream();

input.setReadListener(readListener);
output.setWriteListener(writeListener);
```

Server-Sent Events (SSE) Example

Server-Sent Events allow the server to push updates to the client.

```
@WebServlet(urlPatterns = "/sse", asyncSupported = true)
public class SSEServlet extends HttpServlet {

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

        // Set SSE-related headers
        response.setContentType("text/event-stream");
        response.setCharacterEncoding("UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        response.setHeader("Connection", "keep-alive");

        // Get PrintWriter
        PrintWriter writer = response.getWriter();

        // Start async processing
        final AsyncContext asyncContext = request.startAsync();
        asyncContext.setTimeout(0); // No timeout

        // Create a thread that sends events
        Thread eventThread = new Thread(() -> {
            try {
                int count = 0;
                while (count < 10) { // Send 10 events
                    // Format SSE data
                    String eventData = String.format(
                        "id: %d\n" +
                        "event: message\n" +
                        "data: {\"message\": \"Update %d\", \"time\":
\"%s\"}\n\n",
                        count, count, new java.util.Date());

                    // Write data
                    PrintWriter asyncWriter =
asyncContext.getResponse().getWriter();
                    asyncWriter.write(eventData);
                    asyncWriter.flush();

                    count++;
                    Thread.sleep(2000); // Send an event every 2 seconds
                }

                // Send a final event
                String finalEvent =
                    "id: final\n" +
                    "event: message\n" +
```

```
                    "data: {\"message\": \"All updates complete\", \"time\": \"" +
new java.util.Date() + "\"}\n\n";

                PrintWriter asyncWriter = asyncContext.getResponse().getWriter();
                asyncWriter.write(finalEvent);
                asyncWriter.flush();

                // Complete the async context
                asyncContext.complete();

            } catch (Exception e) {
                e.printStackTrace();
                asyncContext.complete();
            }
        });

        // Start the event thread
        eventThread.start();
    }
}
```

Client-side JavaScript for SSE:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Server-Sent Events Demo</title>
    <script>
      document.addEventListener('DOMContentLoaded', function () {
        var output = document.getElementById('output');

        // Check if SSE is supported
        if (typeof EventSource !== 'undefined') {
          // Create EventSource object
          var source = new EventSource('sse');

          // Event listener for messages
          source.addEventListener(
            'message',
            function (e) {
              var data = JSON.parse(e.data);
              var item = document.createElement('div');
              item.innerHTML =
                '<strong>' + data.time + ':</strong> ' + data.message;
              output.appendChild(item);
            },
            false
          );

          // Event listener for open
          source.addEventListener(
            'open',
```

```
            function (e) {
              var item = document.createElement('div');
              item.innerHTML = 'Connection established';
              output.appendChild(item);
            },
            false
          );

          // Event listener for errors
          source.addEventListener(
            'error',
            function (e) {
              if (e.readyState == EventSource.CLOSED) {
                var item = document.createElement('div');
                item.innerHTML = 'Connection closed';
                output.appendChild(item);
              } else {
                var item = document.createElement('div');
                item.innerHTML = 'Error occurred';
                output.appendChild(item);
              }
            },
            false
          );
        } else {
          output.innerHTML = 'Your browser does not support Server-Sent Events';
        }
      });
    </script>
  </head>
  <body>
    <h1>Server-Sent Events Demo</h1>
    <div id="output"></div>
  </body>
</html>
```

ReadListener and WriteListener Example

For non-blocking I/O operations in Servlet 3.1+:

```
@WebServlet(urlPatterns = "/non-blocking", asyncSupported = true)
public class NonBlockingIOServlet extends HttpServlet {

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

        // Start async processing
        final AsyncContext asyncContext = request.startAsync();

        // Get the output stream
```

```
        ServletOutputStream output = response.getOutputStream();

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

        // Create a write listener for non-blocking writes
        output.setWriteListener(new WriteListener() {
            private final String[] chunks = {
                "<!DOCTYPE html><html><head><title>Non-blocking IO Demo</title>
</head><body>",
                "<h1>Non-blocking IO Demonstration</h1>",
                "<p>This response is being written using non-blocking IO.</p>",
                "<p>Each part of the response is written when the output buffer
has space.</p>",
                "<p>Current time: " + new java.util.Date() + "</p>",
                "</body></html>"
            };

            private int chunkIndex = 0;

            @Override
            public void onWritePossible() throws IOException {
                // Write data while we're able to and have more chunks
                while (output.isReady() && chunkIndex < chunks.length) {
                    output.write(chunks[chunkIndex].getBytes());
                    chunkIndex++;
                }

                // If we're done, complete the async context
                if (chunkIndex >= chunks.length) {
                    asyncContext.complete();
                }
            }

            @Override
            public void onError(Throwable t) {
                t.printStackTrace();
                asyncContext.complete();
            }
        });
    }
}
```

Asynchronous Processing Best Practices

1. **Enable Async Support**: Set `asyncSupported=true` in the `@WebServlet` annotation or `<async-supported>true</async-supported>` in web.xml.
2. **Set Appropriate Timeouts**: Set timeouts based on the expected processing time to avoid resource leaks.
3. **Handle All Exceptions**: Properly handle exceptions in async operations to prevent resource leaks.

4. **Use Thread Pools**: Use appropriate thread pools for async operations rather than creating new threads for each request.
5. **Consider Thread Safety**: Be careful with shared resources in async operations.
6. **Close Resources**: Always close resources (streams, connections) after use.
7. **Test Under Load**: Test async servlets under high load to ensure they perform as expected.
8. **Monitor Performance**: Monitor thread usage and response times to verify benefits.
9. **Use Async Listeners**: Implement AsyncListener to handle events properly.
10. **Ensure Filter Compatibility**: Make sure filters in the chain also support async processing:

```
@WebFilter(urlPatterns = "/*", asyncSupported = true)
public class AsyncSupportFilter implements Filter {
    // Implementation
}
```

2.8 Error Handling

Effective error handling in servlets ensures a good user experience even when problems occur. Java EE provides several mechanisms for handling errors.

Exception Types in Servlets

1. **Checked Exceptions**: Must be caught or declared (e.g., IOException)
2. **Unchecked Exceptions** (RuntimeExceptions): Not required to be caught (e.g., NullPointerException)
3. **Error**: Serious problems that applications should not try to catch (e.g., OutOfMemoryError)

1. Try-Catch Blocks in Servlets

The most basic way to handle exceptions:

```
@WebServlet("/try-catch-demo")
public class TryCatchServlet extends HttpServlet {

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

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>Error Handling Demo</title></head>");
        out.println("<body>");
```

```
        try {
            // Simulating an error condition
            String param = request.getParameter("id");
            int id = Integer.parseInt(param);

            out.println("<h1>Processing ID: " + id + "</h1>");

            // Further processing with the ID
            if (id < 1) {
                throw new IllegalArgumentException("ID must be positive");
            }

            out.println("<p>Operation completed successfully.</p>");

        } catch (NumberFormatException e) {
            // Handle specific exception
            out.println("<h1>Error</h1>");
            out.println("<p>Invalid ID format. Please provide a numeric ID.</p>");

            // Log the exception
            getServletContext().log("NumberFormatException in TryCatchServlet",
e);

        } catch (IllegalArgumentException e) {
            // Handle another specific exception
            out.println("<h1>Error</h1>");
            out.println("<p>Invalid ID value: " + e.getMessage() + "</p>");

            // Log the exception
            getServletContext().log("IllegalArgumentException in TryCatchServlet",
e);

        } catch (Exception e) {
            // Catch-all for unexpected exceptions
            out.println("<h1>Error</h1>");
            out.println("<p>An unexpected error occurred. Please try again later.
</p>");

            // Log the exception
            getServletContext().log("Unexpected error in TryCatchServlet", e);
        }

        out.println("</body>");
        out.println("</html>");
    }
}
```

2. Web.xml Error Pages

Configure custom error pages for specific error codes or exception types:

```
<error-page>
    <error-code>404</error-code>
    <location>/error404.jsp</location>
</error-page>

<error-page>
    <error-code>500</error-code>
    <location>/error500.jsp</location>
</error-page>

<error-page>
    <exception-type>java.lang.NullPointerException</exception-type>
    <location>/errorNullPointer.jsp</location>
</error-page>

<error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/errorServlet.jsp</location>
</error-page>
```

error404.jsp Example:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Page Not Found</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f5f5f5;
        }
        .error-container {
            max-width: 800px;
            margin: 0 auto;
            background-color: #fff;
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        }
        h1 {
            color: #d9534f;
        }
    </style>
</head>
<body>
    <div class="error-container">
```

```
        <h1>404 - Page Not Found</h1>
        <p>The page you are looking for might have been removed, had its name
changed, or is temporarily unavailable.</p>
        <p>Please check the URL for errors or try one of these options:</p>
        <ul>
            <li>Return to the <a href="${pageContext.request.contextPath}/">home
page</a></li>
            <li>Contact <a href="mailto:support@example.com">support</a> for
assistance</li>
        </ul>
        <p>Request URI: ${requestScope['javax.servlet.error.request_uri']}</p>
    </div>
</body>
</html>
```

error500.jsp Example:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Internal Server Error</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f5f5f5;
        }
        .error-container {
            max-width: 800px;
            margin: 0 auto;
            background-color: #fff;
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        }
        h1 {
            color: #d9534f;
        }
        .error-details {
            background-color: #f9f9f9;
            padding: 15px;
            border-radius: 4px;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="error-container">
```

```
        <h1>500 - Internal Server Error</h1>
        <p>Sorry, something went wrong on our server. We're working to fix the
issue.</p>
        <p>Please try again later or contact our support team if the problem
persists.</p>

        <div class="error-details">
            <h3>Error Details (for developers)</h3>
            <p>Timestamp: <%= new java.util.Date() %></p>
            <p>Error Type: ${pageContext.exception.class.name}</p>
            <p>Error Message: ${pageContext.exception.message}</p>
            <p>Request URI: ${requestScope['javax.servlet.error.request_uri']}</p>
            <p>Servlet Name: ${requestScope['javax.servlet.error.servlet_name']}
</p>
        </div>
    </div>
</body>
</html>
```

3. Exception-Specific Error Handling JSP

Example for a JSP that handles NullPointerException:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Null Pointer Exception</title>
</head>
<body>
    <h1>Null Pointer Exception Occurred</h1>
    <p>A null reference was encountered during processing.</p>

    <h2>Error Details:</h2>
    <ul>
        <li>Exception Type: <%= exception.getClass().getName() %></li>
        <li>Error Message: <%= exception.getMessage() %></li>
        <li>Timestamp: <%= new java.util.Date() %></li>
    </ul>

    <p>Stack Trace:</p>
    <pre>
<%
// Print stack trace to a string for display
java.io.StringWriter sw = new java.io.StringWriter();
java.io.PrintWriter pw = new java.io.PrintWriter(sw);
exception.printStackTrace(pw);
out.println(sw.toString());
%>
```

```
    </pre>

    <p><a href="${pageContext.request.contextPath}/">Return to Home Page</a></p>
</body>
</html>
```

4. Programmatic Error Handling

Sending error responses programmatically:

```
@WebServlet("/programmatic-error")
public class ProgrammaticErrorServlet extends HttpServlet {

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

        String action = request.getParameter("action");

        if ("not-found".equals(action)) {
            // Send a 404 error
            response.sendError(HttpServletResponse.SC_NOT_FOUND, "Resource not
found");

        } else if ("forbidden".equals(action)) {
            // Send a 403 error
            response.sendError(HttpServletResponse.SC_FORBIDDEN, "Access denied");

        } else if ("internal-error".equals(action)) {
            // Send a 500 error
            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Internal server error");

        } else if ("redirect".equals(action)) {
            // Redirect to an error page
            response.sendRedirect(request.getContextPath() + "/error.jsp");

        } else if ("exception".equals(action)) {
            // Throw an exception to trigger error page
            throw new ServletException("Intentionally thrown exception for
demonstration");

        } else {
            // Normal response
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head><title>Programmatic Error Handling</title>
</head>");
```

```
            out.println("<body>");
            out.println("<h1>Programmatic Error Handling Demo</h1>");
            out.println("<ul>");
            out.println("<li><a href='?action=not-found'>Trigger 404 Not Found</a>
</li>");
            out.println("<li><a href='?action=forbidden'>Trigger 403 Forbidden</a>
</li>");
            out.println("<li><a href='?action=internal-error'>Trigger 500 Internal
Server Error</a></li>");
            out.println("<li><a href='?action=redirect'>Redirect to Error Page</a>
</li>");
            out.println("<li><a href='?action=exception'>Throw Servlet
Exception</a></li>");
            out.println("</ul>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```

5. Custom Exception Classes

Creating custom exception classes for application-specific errors:

```
// Custom exception class
public class UserNotFoundException extends Exception {

    private int userId;

    public UserNotFoundException(int userId) {
        super("User not found with ID: " + userId);
        this.userId = userId;
    }

    public UserNotFoundException(int userId, String message) {
        super(message);
        this.userId = userId;
    }

    public UserNotFoundException(int userId, String message, Throwable cause) {
        super(message, cause);
        this.userId = userId;
    }

    public int getUserId() {
        return userId;
    }
}
```

Using the custom exception:

```
@WebServlet("/user")
public class UserServlet extends HttpServlet {

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

        String idParam = request.getParameter("id");

        if (idParam == null || idParam.trim().isEmpty()) {
            response.sendError(HttpServletResponse.SC_BAD_REQUEST, "User ID is
required");
            return;
        }

        try {
            int userId = Integer.parseInt(idParam);
            User user = findUser(userId);

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

            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head><title>User Information</title></head>");
            out.println("<body>");
            out.println("<h1>User Information</h1>");
            out.println("<p>ID: " + user.getId() + "</p>");
            out.println("<p>Name: " + user.getName() + "</p>");
            out.println("<p>Email: " + user.getEmail() + "</p>");
            out.println("</body>");
            out.println("</html>");

        } catch (NumberFormatException e) {
            response.sendError(HttpServletResponse.SC_BAD_REQUEST, "Invalid user
ID format");

        } catch (UserNotFoundException e) {
            // Log the exception
            getServletContext().log("User not found", e);

            // Store exception in request for error page
            request.setAttribute("javax.servlet.error.exception", e);
            request.setAttribute("javax.servlet.error.message", e.getMessage());
            request.setAttribute("userId", e.getUserId());

            // Forward to custom error page
            RequestDispatcher dispatcher =
request.getRequestDispatcher("/userNotFound.jsp");
            dispatcher.forward(request, response);
        }
```

```
    }

    private User findUser(int userId) throws UserNotFoundException {
        // In a real application, this would query a database
        // This is a simple simulation
        if (userId == 1) {
            return new User(1, "John Doe", "john@example.com");
        } else if (userId == 2) {
            return new User(2, "Jane Smith", "jane@example.com");
        } else {
            throw new UserNotFoundException(userId);
        }
    }

    // Simple User class for demonstration
    class User {
        private int id;
        private String name;
        private String email;

        public User(int id, String name, String email) {
            this.id = id;
            this.name = name;
            this.email = email;
        }

        public int getId() { return id; }
        public String getName() { return name; }
        public String getEmail() { return email; }
    }
}
```

userNotFound.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>User Not Found</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f5f5f5;
        }
        .error-container {
            max-width: 600px;
            margin: 0 auto;
            background-color: #fff;
```

```
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        }
        h1 {
            color: #d9534f;
        }
    </style>
</head>
<body>
    <div class="error-container">
        <h1>User Not Found</h1>
        <p>${requestScope['javax.servlet.error.message']}</p>
        <p>The user with ID ${userId} could not be found in our system.</p>
        <p>Please check the ID and try again, or contact support if you believe
this is an error.</p>
        <p><a href="${pageContext.request.contextPath}/user?id=1">View Sample
User</a></p>
    </div>
</body>
</html>
```

6. Global Exception Handler

Creating a centralized exception handler:

```
@WebServlet("/error-handler")
public class GlobalExceptionHandler extends HttpServlet {

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

        // Get exception information from request attributes
        Throwable throwable = (Throwable)
request.getAttribute("javax.servlet.error.exception");
        Integer statusCode = (Integer)
request.getAttribute("javax.servlet.error.status_code");
        String servletName = (String)
request.getAttribute("javax.servlet.error.servlet_name");
        String requestUri = (String)
request.getAttribute("javax.servlet.error.request_uri");

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
```

```
        out.println("<title>Error Information</title>");
        out.println("<style>");
        out.println("body { font-family: Arial, sans-serif; margin: 20px; }");
        out.println(".error-container { max-width: 800px; margin: 0 auto; }");
        out.println(".stack-trace { background-color: #f8f9fa; padding: 15px;
border-radius: 4px; overflow-x: auto; }");
        out.println("</style>");
        out.println("</head>");
        out.println("<body>");

        out.println("<div class='error-container'>");
        out.println("<h1>Error Information</h1>");

        out.println("<h2>Basic Information</h2>");
        out.println("<ul>");
        out.println("<li>Status Code: " + (statusCode != null ? statusCode :
"N/A") + "</li>");
        out.println("<li>Request URI: " + (requestUri != null ? requestUri :
"N/A") + "</li>");
        out.println("<li>Servlet Name: " + (servletName != null ? servletName :
"N/A") + "</li>");
        out.println("<li>Error Time: " + new java.util.Date() + "</li>");
        out.println("</ul>");

        if (throwable != null) {
            out.println("<h2>Exception Information</h2>");
            out.println("<ul>");
            out.println("<li>Exception Type: " + throwable.getClass().getName() +
"</li>");
            out.println("<li>Exception Message: " + throwable.getMessage() + "
</li>");
            out.println("</ul>");

            out.println("<h2>Stack Trace</h2>");
            out.println("<div class='stack-trace'>");
            out.println("<pre>");

            // Create a printable stack trace
            StringWriter stringWriter = new StringWriter();
            PrintWriter printWriter = new PrintWriter(stringWriter);
            throwable.printStackTrace(printWriter);
            out.println(stringWriter.toString());

            out.println("</pre>");
            out.println("</div>");
        }

        out.println("<p><a href='" + request.getContextPath() + "'>Return to
Home</a></p>");

        out.println("</div>");
        out.println("</body>");
        out.println("</html>");
```

```
    }
}
```

Configuring the Global Exception Handler in web.xml:

```
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error-handler</location>
</error-page>

<error-page>
    <error-code>404</error-code>
    <location>/error-handler</location>
</error-page>

<error-page>
    <error-code>500</error-code>
    <location>/error-handler</location>
</error-page>
```

Error Handling Best Practices

1. **Create a Consistent Error Handling Strategy**:
    - Define different error pages for different types of errors
    - Use consistent styling and messaging across error pages
    - Provide appropriate user feedback based on error type
2. **Log Errors Appropriately**:
    - Log detailed error information for debugging
    - Include stack traces, request parameters, and user information
    - Consider different log levels for different types of errors
3. **Provide Helpful User Feedback**:
    - Use clear, non-technical language for user-facing error messages
    - Include helpful next steps or alternatives
    - Avoid exposing sensitive information in error messages
4. **Implement Different Strategies for Different Environments**:
    - Show detailed errors in development
    - Show generic, user-friendly errors in production
    - Use environment-specific configuration
5. **Use HTTP Status Codes Correctly**:
    - 400 Bad Request: Invalid input

- 401 Unauthorized: Authentication required
- 403 Forbidden: Authentication succeeded, but access denied
- 404 Not Found: Resource not found
- 500 Internal Server Error: Unexpected server error

6. **Create Custom Exception Classes**:
   - Make exceptions specific to your application domain
   - Include relevant contextual information
   - Follow a consistent naming convention
7. **Centralize Error Handling**:
   - Use a global exception handler for consistency
   - Implement common error handling logic in one place
   - Make it extensible for different types of errors
8. **Handle Asynchronous Errors**:
   - Use AsyncListener.onError() for async servlet errors
   - Set up proper error handling for background threads
9. **Test Error Cases**:
   - Test all error scenarios explicitly
   - Verify that error pages render correctly
   - Ensure error information is properly logged

2.9 Security and Authentication

Security is a critical aspect of web applications. Java EE provides several mechanisms for securing web applications.

1. Authentication Methods

Java EE supports several authentication methods:

1. **BASIC Authentication**: Uses HTTP Basic authentication (username/password)
2. **FORM Authentication**: Custom login form
3. **CLIENT-CERT Authentication**: Client certificate authentication
4. **DIGEST Authentication**: Similar to Basic but encrypts credentials

Configuring Security in web.xml:

```
<security-constraint>
    <display-name>Admin Area</display-name>
    <web-resource-collection>
        <web-resource-name>Admin Resources</web-resource-name>
        <url-pattern>/admin/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
```

```
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>

<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>Admin Realm</realm-name>
    <form-login-config>
        <form-login-page>/login.jsp</form-login-page>
        <form-error-page>/login-error.jsp</form-error-page>
    </form-login-config>
</login-config>

<security-role>
    <role-name>admin</role-name>
</security-role>

<security-role>
    <role-name>user</role-name>
</security-role>
```

Form-Based Authentication Example (login.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Login</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f5f5f5;
        }
        .login-container {
            background-color: #fff;
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            width: 350px;
```

```
        }
        h1 {
            text-align: center;
            margin-bottom: 20px;
        }
        .form-group {
            margin-bottom: 15px;
        }
        label {
            display: block;
            margin-bottom: 5px;
            font-weight: bold;
        }
        input[type="text"], input[type="password"] {
            width: 100%;
            padding: 8px;
            border: 1px solid #ddd;
            border-radius: 4px;
            box-sizing: border-box;
        }
        .submit-btn {
            width: 100%;
            padding: 10px;
            background-color: #4CAF50;
            color: white;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            font-size: 16px;
        }
        .submit-btn:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <div class="login-container">
        <h1>Login</h1>

        <form action="j_security_check" method="post">
            <div class="form-group">
                <label for="j_username">Username:</label>
                <input type="text" id="j_username" name="j_username" required
autofocus>
            </div>

            <div class="form-group">
                <label for="j_password">Password:</label>
                <input type="password" id="j_password" name="j_password" required>
            </div>

            <button type="submit" class="submit-btn">Login</button>
        </form>
    </div>
```

```
</body>
</html>
```

Login Error Page (login-error.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Login Failed</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f5f5f5;
        }
        .error-container {
            background-color: #fff;
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            width: 350px;
            text-align: center;
        }
        h1 {
            color: #d9534f;
        }
        .btn {
            display: inline-block;
            margin-top: 20px;
            padding: 10px 20px;
            background-color: #4CAF50;
            color: white;
            text-decoration: none;
            border-radius: 4px;
        }
    </style>
</head>
<body>
    <div class="error-container">
        <h1>Login Failed</h1>
        <p>Invalid username or password. Please try again.</p>
        <a href="login.jsp" class="btn">Back to Login</a>
    </div>
```

```
</body>
</html>
```

2. Programmatic Security

You can manage security programmatically in servlets:

```
@WebServlet("/security-demo")
public class SecurityDemoServlet extends HttpServlet {

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

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

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>Security Demo</title></head>");
        out.println("<body>");

        out.println("<h1>Security Information</h1>");

        // Get user information
        String remoteUser = request.getRemoteUser();
        boolean isAuthenticated = remoteUser != null;

        out.println("<h2>Authentication Status</h2>");
        if (isAuthenticated) {
            out.println("<p>You are logged in as: " + remoteUser + "</p>");

            Principal userPrincipal = request.getUserPrincipal();
            if (userPrincipal != null) {
                out.println("<p>Principal name: " + userPrincipal.getName() + "
</p>");
            }

            // Check user roles
            out.println("<h2>Role Information</h2>");
            out.println("<ul>");
            out.println("<li>In 'admin' role: " + request.isUserInRole("admin") +
"</li>");
            out.println("<li>In 'user' role: " + request.isUserInRole("user") + "
</li>");
            out.println("</ul>");

            // Logout option
            out.println("<h2>Session Management</h2>");
            out.println("<p><a href='" + response.encodeURL("logout") +
"'>Logout</a></p>");
```

```
        } else {
            out.println("<p>You are not logged in.</p>");
            out.println("<p><a href='login.jsp'>Login</a></p>");
        }

        out.println("</body>");
        out.println("</html>");
    }
}
```

Programmatic Logout Example:

```
@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {

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

        // Invalidate the current session
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.invalidate();
        }

        // Programmatically logout
        request.logout();

        // Redirect to home page
        response.sendRedirect(request.getContextPath() + "/index.jsp");
    }
}
```

3. Custom Authentication with JDBC

If you need more control or want to authenticate against a database:

```
@WebServlet("/custom-login")
public class CustomLoginServlet extends HttpServlet {

    private DataSource dataSource; // Initialize in init() from JNDI

    @Override
    public void init() throws ServletException {
        try {
            Context initContext = new InitialContext();
            Context envContext = (Context) initContext.lookup("java:/comp/env");
            dataSource = (DataSource) envContext.lookup("jdbc/UserDB");
        } catch (NamingException e) {
```

```
            throw new ServletException("Could not locate datasource", e);
        }
    }

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

        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if (username == null || password == null) {
            response.sendRedirect(request.getContextPath() + "/login.jsp?
error=missing");
            return;
        }

        // Authenticate user against database
        try {
            if (authenticateUser(username, password)) {
                // Authentication successful
                HttpSession session = request.getSession();
                session.setAttribute("username", username);

                // Store user roles
                List<String> roles = getUserRoles(username);
                session.setAttribute("userRoles", roles);

                // Redirect to original destination or home page
                String originalURL = (String) session.getAttribute("originalURL");
                if (originalURL != null) {
                    session.removeAttribute("originalURL");
                    response.sendRedirect(originalURL);
                } else {
                    response.sendRedirect(request.getContextPath() +
"/index.jsp");
                }
            } else {
                // Authentication failed
                response.sendRedirect(request.getContextPath() + "/login.jsp?
error=invalid");
            }
        } catch (SQLException e) {
            getServletContext().log("Error during authentication", e);
            response.sendRedirect(request.getContextPath() + "/login.jsp?
error=system");
        }
    }

    private boolean authenticateUser(String username, String password) throws
SQLException {
        String sql = "SELECT password_hash FROM users WHERE username = ?";
```

```
        try (Connection conn = dataSource.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, username);

            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    String storedHash = rs.getString("password_hash");
                    // In a real application, use a secure password hashing
library
                    // This is a simplified example
                    return verifyPassword(password, storedHash);
                }
            }
        }

        return false;
    }

    private boolean verifyPassword(String plainPassword, String storedHash) {
        // In a real application, use a secure password hashing library
        // For example, with BCrypt:
        // return BCrypt.checkpw(plainPassword, storedHash);

        // Simplified example (NOT SECURE for production)
        return storedHash.equals(hashPassword(plainPassword));
    }

    private String hashPassword(String password) {
        // In a real application, use a secure password hashing library
        // For example, with BCrypt:
        // return BCrypt.hashpw(password, BCrypt.gensalt());

        // Simplified example (NOT SECURE for production)
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest(password.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();

            for (byte b : hash) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }

            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Error hashing password", e);
        }
    }

    private List<String> getUserRoles(String username) throws SQLException {
        String sql = "SELECT role FROM user_roles WHERE username = ?";
        List<String> roles = new ArrayList<>();
```

```
        try (Connection conn = dataSource.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, username);

            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    roles.add(rs.getString("role"));
                }
            }
        }

        return roles;
    }
}
```

Custom Authentication Filter:

```
@WebFilter("/*")
public class AuthenticationFilter implements Filter {

    private Set<String> publicURLs;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        publicURLs = new HashSet<>();
        publicURLs.add("/login.jsp");
        publicURLs.add("/custom-login");
        publicURLs.add("/css");
        publicURLs.add("/js");
        publicURLs.add("/images");
        publicURLs.add("/public");
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Get the requested URL
        String requestURI = httpRequest.getRequestURI();
        String contextPath = httpRequest.getContextPath();
        String urlPath = requestURI.substring(contextPath.length());

        // Check if the URL is public
        boolean isPublicResource = false;
        for (String publicURL : publicURLs) {
            if (urlPath.startsWith(publicURL)) {
```

```
                isPublicResource = true;
                break;
            }
        }

        // If public resource, allow access
        if (isPublicResource) {
            chain.doFilter(request, response);
            return;
        }

        // Check if user is authenticated
        HttpSession session = httpRequest.getSession(false);
        boolean isAuthenticated = session != null &&
session.getAttribute("username") != null;

        if (isAuthenticated) {
            // Check role-based access for restricted areas
            if (urlPath.startsWith("/admin")) {
                // Check if user has admin role
                @SuppressWarnings("unchecked")
                List<String> roles = (List<String>)
session.getAttribute("userRoles");
                if (roles != null && roles.contains("admin")) {
                    // User has admin role, allow access
                    chain.doFilter(request, response);
                } else {
                    // User doesn't have admin role, redirect to access denied
page
                    httpResponse.sendRedirect(contextPath + "/access-denied.jsp");
                }
            } else {
                // Not an admin area, allow access
                chain.doFilter(request, response);
            }
        } else {
            // User is not authenticated, save original URL and redirect to login
            session = httpRequest.getSession();
            session.setAttribute("originalURL", requestURI);
            httpResponse.sendRedirect(contextPath + "/login.jsp");
        }
    }

    @Override
    public void destroy() {
        // Cleanup code if needed
    }
}
```

4. HTTPS and Secure Communications

Configuring HTTPS in Tomcat:

1. Generate a Keystore:

```
keytool -genkey -alias tomcat -keyalg RSA -keystore keystore.jks
```

2. Configure Tomcat:

In server.xml:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
           keystoreFile="${catalina.home}/conf/keystore.jks"
keystorePass="changeit"
           clientAuth="false" sslProtocol="TLS" />
```

3. Force HTTPS in web.xml:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Entire Application</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

5. Cross-Site Scripting (XSS) Protection

To protect against XSS attacks, always escape user input:

```
public class XSSFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        chain.doFilter(new XSSRequestWrapper((HttpServletRequest) request),
response);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }
```

```
    @Override
    public void destroy() {
        // Cleanup code
    }

    // Custom request wrapper to sanitize parameters
    private class XSSRequestWrapper extends HttpServletRequestWrapper {

        public XSSRequestWrapper(HttpServletRequest request) {
            super(request);
        }

        @Override
        public String getParameter(String name) {
            String value = super.getParameter(name);
            return value != null ? sanitize(value) : null;
        }

        @Override
        public String[] getParameterValues(String name) {
            String[] values = super.getParameterValues(name);

            if (values == null) {
                return null;
            }

            String[] sanitizedValues = new String[values.length];
            for (int i = 0; i < values.length; i++) {
                sanitizedValues[i] = sanitize(values[i]);
            }

            return sanitizedValues;
        }

        @Override
        public Map<String, String[]> getParameterMap() {
            Map<String, String[]> paramMap = super.getParameterMap();
            Map<String, String[]> sanitizedMap = new HashMap<>();

            for (Map.Entry<String, String[]> entry : paramMap.entrySet()) {
                String[] values = entry.getValue();
                String[] sanitizedValues = new String[values.length];

                for (int i = 0; i < values.length; i++) {
                    sanitizedValues[i] = sanitize(values[i]);
                }

                sanitizedMap.put(entry.getKey(), sanitizedValues);
            }

            return sanitizedMap;
        }

        private String sanitize(String value) {
```

```
            // Replace HTML special characters
            value = value.replaceAll("<", "&lt;")
                         .replaceAll(">", "&gt;")
                         .replaceAll("\"", "&quot;")
                         .replaceAll("'", "&#x27;")
                         .replaceAll("/", "&#x2F;");

            return value;
        }
    }
}
```

6. Cross-Site Request Forgery (CSRF) Protection

Implementing CSRF protection:

```
@WebFilter("/*")
public class CSRFFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Skip CSRF check for GET, HEAD, OPTIONS
        String method = httpRequest.getMethod();
        if (!method.equals("POST") && !method.equals("PUT") &&
!method.equals("DELETE")) {
            // Generate and store CSRF token
            HttpSession session = httpRequest.getSession();
            if (session.getAttribute("csrfToken") == null) {
                String csrfToken = generateCSRFToken();
                session.setAttribute("csrfToken", csrfToken);
            }

            chain.doFilter(request, response);
            return;
        }

        // Check CSRF token for state-changing methods
        HttpSession session = httpRequest.getSession(false);
        if (session != null) {
            String sessionToken = (String) session.getAttribute("csrfToken");
            String requestToken = httpRequest.getParameter("csrfToken");

            if (sessionToken != null && sessionToken.equals(requestToken)) {
                // CSRF token is valid, proceed
                chain.doFilter(request, response);
```

```
            } else {
                // CSRF token is invalid
                httpResponse.sendError(HttpServletResponse.SC_FORBIDDEN, "CSRF
protection validation failed");
            }
        } else {
            // No session or token available
            httpResponse.sendError(HttpServletResponse.SC_FORBIDDEN, "CSRF
protection validation failed");
        }
    }

    private String generateCSRFToken() {
        // Generate a random token
        byte[] bytes = new byte[32];
        new SecureRandom().nextBytes(bytes);
        return Base64.getEncoder().encodeToString(bytes);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

JSP Form with CSRF Protection:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>CSRF Protected Form</title>
</head>
<body>
    <h1>Submit Form</h1>

    <form action="process-form" method="post">
        <div>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>
        </div>

        <div>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
```

```
        </div>

        <!-- CSRF Protection Token -->
        <input type="hidden" name="csrfToken" value="${csrfToken}">

        <div>
            <button type="submit">Submit</button>
        </div>
    </form>
</body>
</html>
```

7. Security Headers

Adding security headers with a filter:

```
@WebFilter("/*")
public class SecurityHeadersFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Add security headers

        // Prevents the browser from MIME-sniffing a response away from the
declared content-type
        httpResponse.setHeader("X-Content-Type-Options", "nosniff");

        // Enables the Cross-site Scripting (XSS) filter in browsers
        httpResponse.setHeader("X-XSS-Protection", "1; mode=block");

        // Prevents the page from being framed (clickjacking protection)
        httpResponse.setHeader("X-Frame-Options", "DENY");

        // HTTP Strict Transport Security (HSTS)
        httpResponse.setHeader("Strict-Transport-Security", "max-age=31536000;
includeSubDomains");

        // Content Security Policy (CSP)
        httpResponse.setHeader("Content-Security-Policy",
                "default-src 'self'; " +
                "script-src 'self' https://ajax.googleapis.com; " +
                "style-src 'self' https://fonts.googleapis.com; " +
                "img-src 'self' data:; " +
                "font-src 'self' https://fonts.gstatic.com; " +
                "connect-src 'self'; " +
                "frame-src 'self'; " +
```

```
                "object-src 'none'");

        // Referrer Policy
        httpResponse.setHeader("Referrer-Policy", "no-referrer-when-downgrade");

        // Feature Policy (now Permissions Policy)
        httpResponse.setHeader("Feature-Policy",
                "geolocation 'self'; " +
                "microphone 'none'; " +
                "camera 'none'");

        // Continue with the filter chain
        chain.doFilter(request, response);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

8. Security Best Practices

1. **Keep Software Updated**:
    - Regular updates for application server, libraries, and dependencies
    - Monitor security bulletins for components you use
2. **Use Secure Communications**:
    - Always use HTTPS for sensitive data
    - Configure TLS properly with modern ciphers
    - Implement HSTS (HTTP Strict Transport Security)
3. **Follow Authentication Best Practices**:
    - Use secure password hashing (bcrypt, Argon2, PBKDF2)
    - Implement account lockout policies
    - Support multi-factor authentication (MFA)
    - Secure session management
4. **Protect Against Common Attacks**:
    - Validate and sanitize all user input
    - Use parameterized queries for database access
    - Implement CSRF protection

- Add security headers
- Protect against XSS, SQL injection, CSRF

5. **Implement Proper Access Control**:

   - Apply principle of least privilege
   - Use role-based access control
   - Check authorization on every sensitive action
   - Verify access control in business logic

6. **Secure Configuration**:

   - Remove default accounts and passwords
   - Disable unnecessary features and components
   - Configure proper error handling
   - Use security-focused server configurations

7. **Audit and Logging**:

   - Log security-relevant events
   - Implement proper log management
   - Include necessary context in logs
   - Protect log integrity

8. **Secure Code Development**:

   - Follow secure coding guidelines
   - Perform security code reviews
   - Use static analysis tools
   - Conduct penetration testing

3. JSP (JavaServer Pages)

3.1 JSP Lifecycle and Architecture

JSP (JavaServer Pages) is a technology that allows developers to create dynamically generated web pages using Java. JSP files are HTML files with embedded Java code.

JSP Architecture

JSPs are part of the Java EE web tier and work alongside servlets:

1. JSP pages are translated into servlet classes by the JSP container
2. The servlet is then compiled and loaded by the servlet container
3. The servlet handles requests and generates responses

JSP technology combines the ease of HTML/XML for presentation with the power of Java for dynamic content generation.

JSP Lifecycle

The lifecycle of a JSP page involves several phases:

1. **Translation Phase**:
    - JSP container converts the JSP page into a servlet source file
    - This happens when the JSP is first requested or when the container starts up (if JSP precompilation is enabled)
2. **Compilation Phase**:
    - The generated servlet source is compiled into a servlet class
    - The compiled class is loaded into the container
3. **Initialization Phase**:
    - The servlet's `jspInit()` method is called
    - Initialization code is executed once when the JSP is loaded
4. **Request Processing Phase**:
    - For each request, the servlet's `_jspService()` method is called
    - The request is processed and the response is generated
5. **Destruction Phase**:
    - The servlet's `jspDestroy()` method is called
    - Cleanup code is executed when the JSP is unloaded

JSP Translation Process

When a JSP is requested for the first time, the container:

1. Parses the JSP page
2. Converts JSP elements to Java code:
    - HTML/Text becomes `out.write()` statements
    - JSP directives become package/import statements
    - JSP declarations become class variables/methods
    - JSP scriptlets become code inside `_jspService()`
    - JSP expressions become `out.print()` statements
3. Creates a servlet class that extends `HttpJspBase` (which extends `HttpServlet`)
4. Compiles the servlet class
5. Loads the compiled class

JSP-Generated Servlet Structure

A simplified version of a generated servlet from a JSP:

```
package org.apache.jsp;

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

```
import javax.servlet.jsp.*;
// Additional imports from JSP directives

public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    // JSP declarations are placed here as class variables/methods

    public void jspInit() {
        // Initialization code
    }

    public void _jspService(HttpServletRequest request, HttpServletResponse
response)
        throws java.io.IOException, ServletException {

        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;
        PageContext _jspx_page_context = null;

        try {
            // Initialize implicit objects
            pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();

            // HTML/Text content becomes out.write() statements
            out.write("<!DOCTYPE html>\n");
            out.write("<html>\n");
            out.write("<head>\n");
            out.write("    <title>Hello JSP</title>\n");
            out.write("</head>\n");
            out.write("<body>\n");

            // JSP scriptlets become regular Java code

            String greeting = "Hello, World!";
            for (int i = 0; i < 3; i++) {

            out.write("\n    <h");
            out.write(String.valueOf(i + 1));
            out.write(">");
            // JSP expressions become out.print() statements
            out.print(greeting);
            out.write("</h");
```

```
                out.write(String.valueOf(i + 1));
                out.write(">\n");

                }

                out.write("\n</body>\n");
                out.write("</html>\n");

            } catch (Throwable t) {
                // Error handling
                if (!(t instanceof SkipPageException)) {
                    out = _jspx_out;
                    if (out != null && out.getBufferSize() != 0)
                        try { out.clearBuffer(); } catch (java.io.IOException e) {}
                    if (_jspx_page_context != null)
    _jspx_page_context.handlePageException(t);
                    else throw new ServletException(t);
                }
            } finally {
                _jspxFactory.releasePageContext(_jspx_page_context);
            }
        }

        public void jspDestroy() {
            // Cleanup code
        }
    }
```

JSP Implicit Objects

JSP provides several predefined objects (implicit objects) that are available in scriptlets and expressions:

Object	Type	Description
request	HttpServletRequest	The client request
response	HttpServletResponse	The response to the client
out	JspWriter	For writing output to the response
session	HttpSession	The user's session (if enabled)
application	ServletContext	The servlet context for the web application
config	ServletConfig	The servlet configuration
pageContext	PageContext	Provides access to various attributes/objects
page	Object (this)	The generated servlet instance (this)
exception	Throwable	Exception object (only in error pages)

Basic JSP Example

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSP Lifecycle Demo</title>
</head>
<body>
    <h1>JSP Lifecycle Demonstration</h1>

    <%-- JSP Declaration (becomes class variable/method) --%>
    <%!
        private int accessCount = 0;

        public void jspInit() {
            System.out.println("JSP Initialized: " + new java.util.Date());
        }

        public void jspDestroy() {
            System.out.println("JSP Destroyed: " + new java.util.Date());
            System.out.println("Total accesses: " + accessCount);
        }

        public String getFormattedDate() {
            return new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new java.util.Date());
        }
    %>

    <%-- JSP Scriptlet (becomes code inside _jspService) --%>
    <%
        accessCount++;
        String userAgent = request.getHeader("User-Agent");
    %>

    <h2>Request Information</h2>
    <ul>
        <li>Current Time: <%= getFormattedDate() %></li>
        <li>Access Count: <%= accessCount %></li>
        <li>User Agent: <%= userAgent %></li>
        <li>Request URI: <%= request.getRequestURI() %></li>
        <li>Server: <%= application.getServerInfo() %></li>
    </ul>

    <h2>JSP Implicit Objects</h2>
    <ul>
        <li>request: <%= request.getClass().getName() %></li>
        <li>response: <%= response.getClass().getName() %></li>
        <li>out: <%= out.getClass().getName() %></li>
        <li>session: <%= session.getClass().getName() %></li>
        <li>application: <%= application.getClass().getName() %></li>
        <li>config: <%= config.getClass().getName() %></li>
```

```
        <li>pageContext: <%= pageContext.getClass().getName() %></li>
        <li>page: <%= page.getClass().getName() %></li>
    </ul>
</body>
</html>
```

Viewing Generated Servlet Code

In Tomcat, you can find the generated servlet classes in:

- `<Tomcat_Install_Dir>/work/Catalina/<hostname>/<webapp>/`

To enable JSP debugging to view the generated code:

- Add `development="true"` to the JSP servlet configuration in web.xml
- Check the server logs for the location of the generated files

JSP Precompilation

Precompiling JSPs offers several advantages:

- Faster initial response times for users
- Early detection of JSP errors
- Simplified deployment

How to Precompile JSPs:

1. Using the `jspc` tool (included with Tomcat):

```
jspc -webapp <webapp-dir> -d <output-dir>
```

2. Using Maven:

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <jspc>
            <verbose>true</verbose>
            <source>1.8</source>
            <target>1.8</target>
        </jspc>
    </configuration>
</plugin>
```

Performance Considerations

1. **Buffer Size**:
    - Control output buffering with the `buffer` attribute in the page directive
    - Larger buffers can improve performance for larger pages
2. **Auto-Flush**:
    - Control when the buffer is flushed with the `autoFlush` attribute
    - Set to false to prevent automatic flushing (throws exception when buffer overflow occurs)
3. **Thread Safety**:
    - Be careful with declarations (class variables) in multi-threaded environments
    - Avoid instance variables unless they are thread-safe or read-only
4. **Error Handling**:
    - Use errorPage attribute to specify an error page
    - Implement proper exception handling
5. **Include vs. Forward**:
    - Choose the appropriate mechanism for code reuse
    - Include adds content at request time
    - Forward transfers control to another resource

3.2 Scriptlets, Declarations, Expressions

JSP provides several scripting elements to embed Java code within HTML pages:

1. **Declarations**: Define methods or fields in the generated servlet class
2. **Scriptlets**: Insert Java code in the _jspService method
3. **Expressions**: Output the result of a Java expression directly to the page

JSP Declarations

Declarations define variables and methods at the class level in the generated servlet:

```
<%!
    // Variables declared here become instance variables (fields) of the servlet
    private int counter = 0;
    private final String appName = "My JSP Application";
    private final java.util.Date startTime = new java.util.Date();

    // Methods declared here become instance methods of the servlet
    public int incrementCounter() {
        return ++counter;
    }

    public String getFormattedStartTime() {
        return new java.text.SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(startTime);
```

```
    }

    // You can override the jspInit and jspDestroy methods
    public void jspInit() {
        // Initialization code
        System.out.println("JSP initialized at " + new java.util.Date());
    }

    public void jspDestroy() {
        // Cleanup code
        System.out.println("JSP destroyed at " + new java.util.Date());
    }
%>
```

Key Points about Declarations:

- Declarations are placed at the class level, outside the service method
- Variables declared are instance variables (shared across requests)
- Methods declared are instance methods
- Code in declarations executes only once when the class is loaded
- Be careful with thread safety since variables are shared across requests

JSP Scriptlets

Scriptlets contain Java code that executes within the _jspService method:

```
<%
    // Local variables - created for each request
    String userName = request.getParameter("name");
    if (userName == null || userName.trim().isEmpty()) {
        userName = "Guest";
    }

    // Conditional logic
    if (session.isNew()) {
        session.setAttribute("visitCount", 1);
        out.println("<p>Welcome, new visitor!</p>");
    } else {
        int visitCount = (Integer) session.getAttribute("visitCount");
        session.setAttribute("visitCount", visitCount + 1);
        out.println("<p>Welcome back! Visit count: " + visitCount + "</p>");
    }

    // Loops
    out.println("<ul>");
    for (int i = 1; i <= 5; i++) {
        out.println("<li>Item " + i + "</li>");
    }
    out.println("</ul>");
%>
```

Key Points about Scriptlets:

- Code in scriptlets executes each time the JSP is requested
- Variables declared are local to the service method (not shared across requests)
- You can access the implicit objects (request, response, session, etc.)
- Scriptlets can be intermixed with HTML and other JSP elements
- Multiple scriptlets in a JSP are treated as part of the same method (_jspService)

JSP Expressions

Expressions output the result of a Java expression directly to the page:

```
<p>Current time: <%= new java.util.Date() %></p>
<p>Server: <%= application.getServerInfo() %></p>
<p>Session ID: <%= session.getId() %></p>
<p>User's IP: <%= request.getRemoteAddr() %></p>
<p>Counter value: <%= incrementCounter() %></p>
<p>Application uptime: <%= System.currentTimeMillis() - startTime.getTime() %>
ms</p>
```

Key Points about Expressions:

- Expressions are evaluated at request time
- Result is converted to a string and inserted into the output
- No semicolon at the end of the expression
- Equivalent to `out.print(expression)`
- Can call methods, access variables, and use operators

Combining JSP Elements

A complete example combining declarations, scriptlets, and expressions:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSP Scripting Elements Demo</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .counter { font-weight: bold; color: #007bff; }
        .user-info { background-color: #f8f9fa; padding: 10px; margin: 10px 0;
border-radius: 5px; }
        .timestamp { font-size: 0.8em; color: #6c757d; }
    </style>
</head>
<body>
```

```
    <%-- Declaration: Class-level fields and methods --%>
    <%!
        private int visitorCounter = 0;
        private final java.util.List<String> recentVisitors = new
java.util.ArrayList<>();

        public synchronized int incrementAndGetCounter() {
            return ++visitorCounter;
        }

        public synchronized void addVisitor(String ipAddress) {
            // Keep only the last 5 visitors
            if (recentVisitors.size() >= 5) {
                recentVisitors.remove(0);
            }
            recentVisitors.add(ipAddress + " at " + new java.util.Date());
        }
    %>

    <%-- Scriptlet: Request processing logic --%>
    <%
        // Get user information
        String userAgent = request.getHeader("User-Agent");
        String ipAddress = request.getRemoteAddr();
        String requestURI = request.getRequestURI();

        // Update visitor counter and list
        int visitorNumber = incrementAndGetCounter();
        addVisitor(ipAddress);

        // Set a session attribute for demonstration
        String userName = request.getParameter("name");
        if (userName != null && !userName.trim().isEmpty()) {
            session.setAttribute("userName", userName);
        } else {
            userName = (String) session.getAttribute("userName");
            if (userName == null) {
                userName = "Guest";
            }
        }
    %>

    <h1>JSP Scripting Elements Demonstration</h1>

    <div class="user-info">
        <h2>Welcome, <%= userName %>!</h2>
        <p>You are visitor number <span class="counter"><%= visitorNumber %>
</span></p>
        <p>Your IP address: <%= ipAddress %></p>
        <p>You are using: <%= userAgent %></p>
        <p>The current time is: <%= new java.util.Date() %></p>
    </div>

    <h2>Recent Visitors</h2>
```

```
    <ul>
        <%-- Scriptlet with loop structure --%>
        <%
            for (String visitor : recentVisitors) {
        %>
            <li><%= visitor %></li>
        <%
            }
        %>
    </ul>

    <h2>Session Information</h2>
    <p>Session ID: <%= session.getId() %></p>
    <p>Session Creation Time: <%= new java.util.Date(session.getCreationTime()) %>
</p>
    <p>Last Accessed Time: <%= new java.util.Date(session.getLastAccessedTime())
%></p>

    <h2>Application Information</h2>
    <p>Server Info: <%= application.getServerInfo() %></p>
    <p>Servlet API Version: <%= application.getMajorVersion() %>.<%=
application.getMinorVersion() %></p>

    <form action="" method="get">
        <div>
            <label for="name">Your Name:</label>
            <input type="text" id="name" name="name" value="<%= userName %>">
            <button type="submit">Update</button>
        </div>
    </form>

    <p class="timestamp">Page generated at: <%= new java.util.Date() %></p>
</body>
</html>
```

Best Practices for JSP Scripting Elements

1. **Minimize Java Code in JSPs**:
    - Separate presentation from business logic
    - Move complex logic to beans or helper classes
    - Use JSP tags and Expression Language instead of scriptlets
2. **Thread Safety**:
    - Be careful with declarations (class-level variables)
    - Use synchronization for shared resources
    - Consider making methods and variables thread-safe
3. **Code Organization**:
    - Group related declarations together

- Place declarations at the top of the JSP
- Use comments to document code
- Split complex JSPs into smaller ones with includes

4. **Error Handling**:
   - Use try-catch blocks in scriptlets to handle exceptions
   - Set up error pages using the page directive
5. **Performance Optimization**:
   - Minimize the use of expressions in loops
   - Precompute values when possible
   - Buffer output appropriately
6. **Modern Alternatives**:
   - Consider JSTL (JSP Standard Tag Library) instead of scriptlets
   - Use EL (Expression Language) instead of expressions
   - Look into tag files for reusable components

3.3 Directives, Actions, and Implicit Objects

JSP Directives provide instructions to the JSP container about how to process the page. Actions allow for dynamic content inclusion and bean manipulation. Implicit objects are automatically available for use in JSP pages.

JSP Directives

Directives affect the overall structure of the servlet class. There are three types of directives:

1. **Page Directive**: Controls page attributes
2. **Include Directive**: Includes content at translation time
3. **Taglib Directive**: Declares custom tag libraries

Page Directive

The page directive defines page-specific attributes:

```
<%@ page language="java"
         contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8"
         import="java.util.*, java.text.SimpleDateFormat"
         session="true"
         buffer="8kb"
         autoFlush="true"
         errorPage="error.jsp"
         isErrorPage="false"
         isELIgnored="false"
         isThreadSafe="true" %>
```

Common Page Directive Attributes:

Attribute	Description	Example
language	Scripting language used	language="java"
contentType	MIME type and character encoding	contentType="text/html; charset=UTF-8"
pageEncoding	Character encoding for the page	pageEncoding="UTF-8"
import	Java classes to import	import="java.util.*, java.text.*"
session	Whether page uses HTTP sessions	session="true"
buffer	Output buffer size	buffer="16kb"
autoFlush	Whether buffer should auto-flush	autoFlush="true"
errorPage	URL of error handling page	errorPage="error.jsp"
isErrorPage	Whether this page is an error page	isErrorPage="true"
isELIgnored	Whether EL expressions are ignored	isELIgnored="false"
isThreadSafe	Whether page is thread-safe	isThreadSafe="true"
info	Information about the page	info="User registration page"
trimDirectiveWhitespaces	Remove whitespace from directives	trimDirectiveWhitespaces="true"

Example of a Page with Multiple Page Directives:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ page import="java.util.*" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="com.example.User" %>
<%@ page errorPage="errorHandler.jsp" %>
<%@ page session="true" buffer="16kb" autoFlush="true" %>

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Page Directive Demo</title>
</head>
<body>
    <h1>Page Directive Demonstration</h1>

    <%
        // Using imported classes
        Date now = new Date();
```

```
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String formattedDate = sdf.format(now);

        // Create a list for demonstration
        List<String> items = new ArrayList<>();
        items.add("Item 1");
        items.add("Item 2");
        items.add("Item 3");

        // Access session (enabled by session="true")
        Integer visitCount = (Integer) session.getAttribute("visitCount");
        if (visitCount == null) {
            visitCount = 1;
        } else {
            visitCount++;
        }
        session.setAttribute("visitCount", visitCount);

        // Demonstrate buffer
        for (int i = 0; i < 100; i++) {
            out.print(" "); // Add space to fill buffer
        }

        // Uncomment to test error page
        // int result = 10 / 0; // Will trigger error page
    %>

    <p>Current Date and Time: <%= formattedDate %></p>

    <h2>List Items:</h2>
    <ul>
        <% for (String item : items) { %>
            <li><%= item %></li>
        <% } %>
    </ul>

    <p>Visit Count: <%= visitCount %></p>
    <p>Buffer Size: <%= out.getBufferSize() %> bytes</p>
    <p>Remaining in Buffer: <%= out.getRemaining() %> bytes</p>
    <p>Auto-Flush Enabled: <%= out.isAutoFlush() %></p>
</body>
</html>
```

Include Directive

The include directive includes content from another file at translation time (static include):

```
<%@ include file="header.jsp" %>
<div class="content">
    <h1>Main Content</h1>
    <p>This is the main content of the page.</p>
```

```
</div>
<%@ include file="footer.jsp" %>
```

Key Points about Include Directive:

- Inclusion happens during translation, before the JSP is compiled
- Included content becomes part of the original JSP
- Variables and methods from included files are accessible
- Good for common elements like headers, footers, and navigation
- Changes in included files require recompilation of the including JSP

Example of header.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>My Website - <%= request.getParameter("pageTitle") %></title>
    <link rel="stylesheet" href="styles.css">
    <script src="scripts.js"></script>
</head>
<body>
    <header>
        <h1 class="logo">My Website</h1>
        <nav>
            <ul>
                <li><a href="index.jsp">Home</a></li>
                <li><a href="about.jsp">About</a></li>
                <li><a href="services.jsp">Services</a></li>
                <li><a href="contact.jsp">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
```

Example of footer.jsp:

```
    </main>
    <footer>
        <p>&copy; <%= new java.util.Date().getYear() + 1900 %> My Website. All
rights reserved.</p>
        <p>Last updated: <%= new java.text.SimpleDateFormat("yyyy-MM-
dd").format(new java.util.Date()) %></p>
    </footer>
</body>
</html>
```

Taglib Directive

The taglib directive declares a tag library and the prefix used to access it:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

Example Using Tag Libraries:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Tag Library Demo</title>
</head>
<body>
    <h1>JSTL Tag Library Demonstration</h1>

    <%-- Set variables --%>
    <c:set var="now" value="<%= new java.util.Date() %>" />
    <c:set var="salary" value="5000" />

    <h2>Formatted Date:</h2>
    <p>Short Date: <fmt:formatDate value="${now}" type="date" dateStyle="short" />
</p>
    <p>Long Date: <fmt:formatDate value="${now}" type="date" dateStyle="long" />
</p>
    <p>Full Date and Time: <fmt:formatDate value="${now}" type="both"
dateStyle="full" timeStyle="full" /></p>

    <h2>Formatted Number:</h2>
    <p>Currency: <fmt:formatNumber value="${salary}" type="currency" /></p>

    <h2>Conditional Logic:</h2>
    <c:if test="${salary > 4000}">
        <p>Your salary is good.</p>
    </c:if>

    <c:choose>
        <c:when test="${salary <= 3000}">
            <p>Low salary.</p>
        </c:when>
        <c:when test="${salary > 3000 && salary <= 6000}">
            <p>Medium salary.</p>
```

```
            </c:when>
            <c:otherwise>
                <p>High salary.</p>
            </c:otherwise>
        </c:choose>

        <h2>Iteration:</h2>
        <ul>
            <c:forEach var="i" begin="1" end="5">
                <li>Item ${i}</li>
            </c:forEach>
        </ul>
    </body>
    </html>
```

JSP Actions

JSP actions use XML syntax to control the behavior of the servlet engine. They allow for dynamic content inclusion and bean manipulation.

Common JSP actions include:

1. **jsp:include**: Includes content at request time
2. **jsp:forward**: Forwards request to another resource
3. **jsp:param**: Adds parameters to an include or forward
4. **jsp:useBean**: Creates or locates a JavaBean
5. **jsp:setProperty**: Sets a property of a JavaBean
6. **jsp:getProperty**: Gets a property of a JavaBean
7. **jsp:plugin**: Generates browser-specific code for applets/objects

jsp:include Action

The jsp:include action includes the content of another resource at request time (dynamic include):

```
<jsp:include page="header.jsp">
    <jsp:param name="pageTitle" value="Home Page" />
</jsp:include>

<div class="content">
    <h1>Welcome to Our Website</h1>
    <p>This is the main content of the page.</p>
</div>

<jsp:include page="footer.jsp" />
```

Key Points about jsp:include:

- Inclusion happens at request time, not translation time
- Included content is processed separately and result is included

- Variables and methods from included files are not accessible
- Good for dynamic content that changes frequently
- Changes in included files don't require recompilation of including JSP

jsp:forward Action

The jsp:forward action forwards the request to another resource:

```
<%
    String userType = request.getParameter("userType");
    if (userType != null) {
        if (userType.equals("admin")) {
%>
            <jsp:forward page="admin.jsp">
                <jsp:param name="source" value="main" />
            </jsp:forward>
<%
        } else if (userType.equals("user")) {
%>
            <jsp:forward page="user.jsp">
                <jsp:param name="source" value="main" />
            </jsp:forward>
<%
        }
    }
%>

<h1>Main Page</h1>
<p>Please select your user type:</p>
<a href="?userType=admin">Admin</a> | <a href="?userType=user">User</a>
```

Key Points about jsp:forward:

- Control is completely transferred to the target resource
- The URL in the browser remains unchanged
- The original JSP's output buffer is cleared
- Request attributes are preserved
- Can add parameters with jsp:param

jsp:useBean, jsp:setProperty, jsp:getProperty Actions

These actions work with JavaBeans to separate business logic from presentation:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
```

```
    <title>Bean Demo</title>
</head>
<body>
    <h1>JavaBean Demonstration</h1>

    <%-- Create or locate a JavaBean --%>
    <jsp:useBean id="user" class="com.example.User" scope="session">
        <%-- This code runs only if the bean is created for the first time --%>
        <jsp:setProperty name="user" property="firstName" value="John" />
        <jsp:setProperty name="user" property="lastName" value="Doe" />
    </jsp:useBean>

    <%-- Set properties from request parameters --%>
    <jsp:setProperty name="user" property="*" />

    <%-- Set a specific property --%>
    <jsp:setProperty name="user" property="email" value="john.doe@example.com" />

    <h2>User Information:</h2>
    <p>First Name: <jsp:getProperty name="user" property="firstName" /></p>
    <p>Last Name: <jsp:getProperty name="user" property="lastName" /></p>
    <p>Email: <jsp:getProperty name="user" property="email" /></p>
    <p>Full Name: <jsp:getProperty name="user" property="fullName" /></p>

    <h2>Update User Information:</h2>
    <form action="" method="post">
        <div>
            <label for="firstName">First Name:</label>
            <input type="text" id="firstName" name="firstName" value="
<jsp:getProperty name="user" property="firstName" />">
        </div>
        <div>
            <label for="lastName">Last Name:</label>
            <input type="text" id="lastName" name="lastName" value="
<jsp:getProperty name="user" property="lastName" />">
        </div>
        <div>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" value="<jsp:getProperty
name="user" property="email" />">
        </div>
        <div>
            <button type="submit">Update</button>
        </div>
    </form>
</body>
</html>
```

User.java (JavaBean):

```
package com.example;
```

```
import java.io.Serializable;

public class User implements Serializable {
    private static final long serialVersionUID = 1L;

    private String firstName;
    private String lastName;
    private String email;

    // Default constructor (required for JavaBeans)
    public User() {
    }

    // Getters and setters
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    // Additional getter for full name
    public String getFullName() {
        return firstName + " " + lastName;
    }
}
```

Key Points about Bean Actions:

- `jsp:useBean` creates a new bean or locates an existing one
- `scope` attribute specifies the scope (page, request, session, application)
- `jsp:setProperty` sets bean properties (with `property="*"` to set all from request)
- `jsp:getProperty` gets bean property values
- JavaBeans must follow specific conventions (default constructor, getter/setter methods)

JSP Implicit Objects

JSP provides several predefined objects that are automatically available in scriptlets and expressions:

Object	Type	Description
request	HttpServletRequest	The client request
response	HttpServletResponse	The response to the client
out	JspWriter	For writing output to the response
session	HttpSession	The user's session (if enabled)
application	ServletContext	The servlet context for the web application
config	ServletConfig	The servlet configuration
pageContext	PageContext	Provides access to various attributes/objects
page	Object (this)	The generated servlet instance (this)
exception	Throwable	Exception object (only in error pages)

Example Using Implicit Objects:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Implicit Objects Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>JSP Implicit Objects Demonstration</h1>

    <h2>Request Information</h2>
    <table>
        <tr><th>Property</th><th>Value</th></tr>
        <tr><td>Method</td><td><%= request.getMethod() %></td></tr>
        <tr><td>Request URI</td><td><%= request.getRequestURI() %></td></tr>
        <tr><td>Protocol</td><td><%= request.getProtocol() %></td></tr>
        <tr><td>Context Path</td><td><%= request.getContextPath() %></td></tr>
        <tr><td>Servlet Path</td><td><%= request.getServletPath() %></td></tr>
        <tr><td>Query String</td><td><%= request.getQueryString() %></td></tr>
        <tr><td>Remote Address</td><td><%= request.getRemoteAddr() %></td></tr>
        <tr><td>Remote Host</td><td><%= request.getRemoteHost() %></td></tr>
        <tr><td>User Agent</td><td><%= request.getHeader("User-Agent") %></td>
```

```
</tr>
    </table>

    <h2>Session Information</h2>
    <table>
        <tr><th>Property</th><th>Value</th></tr>
        <tr><td>Session ID</td><td><%= session.getId() %></td></tr>
        <tr><td>Creation Time</td><td><%= new
java.util.Date(session.getCreationTime()) %></td></tr>
        <tr><td>Last Accessed Time</td><td><%= new
java.util.Date(session.getLastAccessedTime()) %></td></tr>
        <tr><td>Max Inactive Interval</td><td><%= session.getMaxInactiveInterval()
%> seconds</td></tr>
        <tr><td>New Session?</td><td><%= session.isNew() %></td></tr>
    </table>

    <h2>Application Information</h2>
    <table>
        <tr><th>Property</th><th>Value</th></tr>
        <tr><td>Server Info</td><td><%= application.getServerInfo() %></td></tr>
        <tr><td>Servlet API Version</td><td><%= application.getMajorVersion() %>.
<%= application.getMinorVersion() %></td></tr>
        <tr><td>Real Path</td><td><%= application.getRealPath("/") %></td></tr>
    </table>

    <h2>Page and Config Information</h2>
    <table>
        <tr><th>Property</th><th>Value</th></tr>
        <tr><td>Page Class</td><td><%= page.getClass().getName() %></td></tr>
        <tr><td>Servlet Name</td><td><%= config.getServletName() %></td></tr>
    </table>

    <h2>Output Information</h2>
    <table>
        <tr><th>Property</th><th>Value</th></tr>
        <tr><td>Buffer Size</td><td><%= out.getBufferSize() %> bytes</td></tr>
        <tr><td>Remaining in Buffer</td><td><%= out.getRemaining() %> bytes</td>
</tr>
        <tr><td>Auto-Flush</td><td><%= out.isAutoFlush() %></td></tr>
    </table>

    <h2>PageContext Information</h2>
    <table>
        <tr><th>Property</th><th>Value</th></tr>
        <tr><td>Error Data?</td><td><%= pageContext.getErrorData() != null %></td>
</tr>
        <tr><td>Page Scope Attributes</td><td>
            <%
                java.util.Enumeration<String> attrs =
pageContext.getAttributeNamesInScope(PageContext.PAGE_SCOPE);
                while (attrs.hasMoreElements()) {
                    out.print(attrs.nextElement());
                    if (attrs.hasMoreElements()) out.print(", ");
                }
```

```
            %>
        </td></tr>
    </table>

    <% pageContext.setAttribute("testAttribute", "Test Value",
PageContext.PAGE_SCOPE); %>
    <p>Setting an attribute in PageContext: <%=
pageContext.getAttribute("testAttribute") %></p>
</body>
</html>
```

Comparing Directives and Actions

Feature	Directives	Actions
Syntax	`<%@ directive attribute="value" %>`	`<jsp:action attribute="value" />`
Processing Time	Translation time	Request time
Purpose	Provide instructions to the JSP container	Perform dynamic operations during request
Common Uses	Page configuration, imports, tag libraries	Include content, bean manipulation
Effect on Generated Servlet	Affects structure of the servlet class	Translated to method calls in _jspService
Example of Static Include	`<%@ include file="header.jsp" %>`	N/A
Example of Dynamic Include	N/A	`<jsp:include page="header.jsp" />`

3.4 Expression Language (EL)

The JSP Expression Language (EL) provides a simple and concise way to access data stored in JavaBeans, collections, and implicit objects without using Java code in JSP.

Introduction to Expression Language

EL was introduced to simplify JSP code and reduce the use of scriptlets. It follows the syntax `${expression}` and is primarily used to:

1. Access bean properties
2. Access collection values
3. Access implicit objects
4. Perform operations (arithmetic, logical, etc.)

Basic EL Syntax

EL expressions are enclosed in ${ and }:

```
<p>User name: ${user.name}</p>
<p>First item: ${items[0]}</p>
<p>Request parameter: ${param.id}</p>
<p>Session attribute: ${sessionScope.count}</p>
<p>Calculation: ${2 * 3 + 4}</p>
```

Starting with JSP 2.1, you can also use EL expressions in template text (outside tags) with the syntax #{expression}:

```
<p>User name: #{user.name}</p>
```

EL Variables and Scopes

EL can access variables from different scopes:

```
<%-- Set some attributes in different scopes --%>
<%
    pageContext.setAttribute("pageMessage", "Page scope message");
    request.setAttribute("requestMessage", "Request scope message");
    session.setAttribute("sessionMessage", "Session scope message");
    application.setAttribute("applicationMessage", "Application scope message");

    // Also set a variable with the same name in different scopes
    pageContext.setAttribute("message", "Page message");
    request.setAttribute("message", "Request message");
    session.setAttribute("message", "Session message");
    application.setAttribute("message", "Application message");
%>

<%-- Access attributes from specific scopes --%>
<h2>Accessing Attributes from Specific Scopes</h2>
<p>Page scope: ${pageScope.pageMessage}</p>
<p>Request scope: ${requestScope.requestMessage}</p>
<p>Session scope: ${sessionScope.sessionMessage}</p>
<p>Application scope: ${applicationScope.applicationMessage}</p>

<%-- Access an attribute without specifying scope (searches all scopes) --%>
<h2>Accessing Attributes Without Specifying Scope</h2>
<p>Generic access: ${message}</p>
<p>This finds the first match in this order: page, request, session,
application</p>
<p>So it returns: ${pageScope.message}</p>
```

EL Implicit Objects

EL provides several implicit objects that can be used in expressions:

Object	Description
pageScope	Map of page-scoped attributes
requestScope	Map of request-scoped attributes
sessionScope	Map of session-scoped attributes
applicationScope	Map of application-scoped attributes
param	Map of request parameters (single values)
paramValues	Map of request parameters (array of values)
header	Map of request headers (single values)
headerValues	Map of request headers (array of values)
cookie	Map of cookies
initParam	Map of context initialization parameters
pageContext	The PageContext object

Example Using EL Implicit Objects:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>EL Implicit Objects Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>EL Implicit Objects Demonstration</h1>

    <h2>Request Parameters</h2>
    <p>Access with param and paramValues:</p>
    <p>Name parameter: ${param.name}</p>
    <p>Multiple values for 'hobby' parameter:</p>
    <ul>
        <li>First hobby: ${paramValues.hobby[0]}</li>
        <li>Second hobby: ${paramValues.hobby[1]}</li>
    </ul>

    <h2>Request Headers</h2>
    <p>User-Agent: ${header['User-Agent']}</p>
```

```
    <p>Accept-Language: ${header['Accept-Language']}</p>

    <h2>Cookies</h2>
    <p>JSESSIONID cookie: ${cookie.JSESSIONID.value}</p>

    <h2>Context Init Parameters</h2>
    <p>Database URL: ${initParam.dbURL}</p>

    <h2>PageContext Object</h2>
    <p>Context Path: ${pageContext.request.contextPath}</p>
    <p>Request Method: ${pageContext.request.method}</p>
    <p>Session ID: ${pageContext.session.id}</p>
    <p>Server Info: ${pageContext.servletContext.serverInfo}</p>

    <h2>Try It Out:</h2>
    <form action="" method="get">
        <div>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" value="${param.name}">
        </div>
        <div>
            <label>Hobbies:</label>
            <input type="checkbox" id="hobby1" name="hobby" value="reading">
            <label for="hobby1">Reading</label>
            <input type="checkbox" id="hobby2" name="hobby" value="sports">
            <label for="hobby2">Sports</label>
            <input type="checkbox" id="hobby3" name="hobby" value="music">
            <label for="hobby3">Music</label>
        </div>
        <div>
            <button type="submit">Submit</button>
        </div>
    </form>
</body>
</html>
```

Accessing JavaBean Properties with EL

EL provides a simple way to access bean properties:

```
<%-- Create a User bean --%>
<jsp:useBean id="user" class="com.example.User" scope="session">
    <jsp:setProperty name="user" property="firstName" value="John" />
    <jsp:setProperty name="user" property="lastName" value="Doe" />
    <jsp:setProperty name="user" property="email" value="john.doe@example.com" />
</jsp:useBean>

<%-- Access bean properties with EL --%>
<h2>User Information (Using EL)</h2>
<p>First Name: ${user.firstName}</p>
<p>Last Name: ${user.lastName}</p>
```

```
<p>Email: ${user.email}</p>
<p>Full Name: ${user.fullName}</p>
```

Accessing Nested Properties:

```
<%-- Create an Address bean and set it in the User bean --%>
<jsp:useBean id="address" class="com.example.Address" scope="session">
    <jsp:setProperty name="address" property="street" value="123 Main St" />
    <jsp:setProperty name="address" property="city" value="Springfield" />
    <jsp:setProperty name="address" property="state" value="IL" />
    <jsp:setProperty name="address" property="zipCode" value="12345" />
</jsp:useBean>

<%
    user.setAddress(address);
%>

<%-- Access nested properties with EL --%>
<h2>User Address (Using EL)</h2>
<p>Street: ${user.address.street}</p>
<p>City: ${user.address.city}</p>
<p>State: ${user.address.state}</p>
<p>ZIP Code: ${user.address.zipCode}</p>
<p>Full Address: ${user.address.fullAddress}</p>
```

Accessing Collections with EL

EL can access data in arrays, lists, and maps:

```
<%-- Create some collections --%>
<%
    // Array
    String[] colors = {"Red", "Green", "Blue", "Yellow", "Purple"};
    pageContext.setAttribute("colors", colors);

    // List
    java.util.List<String> fruits = new java.util.ArrayList<>();
    fruits.add("Apple");
    fruits.add("Banana");
    fruits.add("Orange");
    fruits.add("Mango");
    pageContext.setAttribute("fruits", fruits);

    // Map
    java.util.Map<String, String> countries = new java.util.HashMap<>();
    countries.put("US", "United States");
    countries.put("UK", "United Kingdom");
    countries.put("CA", "Canada");
    countries.put("AU", "Australia");
```

```
    pageContext.setAttribute("countries", countries);
%>

<%-- Access array elements --%>
<h2>Colors (Array)</h2>
<ul>
    <li>First color: ${colors[0]}</li>
    <li>Second color: ${colors[1]}</li>
    <li>Third color: ${colors[2]}</li>
    <li>Array length: ${colors.length}</li>
</ul>

<%-- Access list elements --%>
<h2>Fruits (List)</h2>
<ul>
    <li>First fruit: ${fruits[0]}</li>
    <li>Second fruit: ${fruits[1]}</li>
    <li>Third fruit: ${fruits[2]}</li>
    <li>List size: ${fruits.size()}</li>
</ul>

<%-- Access map elements --%>
<h2>Countries (Map)</h2>
<ul>
    <li>US: ${countries.US}</li>
    <li>UK: ${countries.UK}</li>
    <li>CA: ${countries["CA"]}</li>
    <li>AU: ${countries["AU"]}</li>
    <li>Map size: ${countries.size()}</li>
</ul>
```

EL Operators

EL supports various operators for arithmetic, logical, and relational operations:

1. **Arithmetic Operators**: `+`, `-`, `*`, `/`, `%`
2. **Logical Operators**: `&&` (and), `||` (or), `!` (not)
3. **Relational Operators**: `==`, `!=`, `<`, `>`, `<=`, `>=`
4. **Empty Operator**: `empty` (checks if a value is null or empty)
5. **Conditional Operator**: `? :` (ternary operator)

Example Using EL Operators:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>EL Operators Demo</title>
    <style>
```

```
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>EL Operators Demonstration</h1>

    <%-- Set up some variables --%>
    <%
        pageContext.setAttribute("x", 10);
        pageContext.setAttribute("y", 5);
        pageContext.setAttribute("name", "John");
        pageContext.setAttribute("emptyString", "");
        pageContext.setAttribute("nullValue", null);
        java.util.List<String> emptyList = new java.util.ArrayList<>();
        pageContext.setAttribute("emptyList", emptyList);
        java.util.List<String> nonEmptyList = new java.util.ArrayList<>();
        nonEmptyList.add("Item");
        pageContext.setAttribute("nonEmptyList", nonEmptyList);
    %>

    <h2>Arithmetic Operators</h2>
    <table>
        <tr><th>Expression</th><th>Result</th></tr>
        <tr><td>${x} + ${y}</td><td>${x + y}</td></tr>
        <tr><td>${x} - ${y}</td><td>${x - y}</td></tr>
        <tr><td>${x} * ${y}</td><td>${x * y}</td></tr>
        <tr><td>${x} / ${y}</td><td>${x / y}</td></tr>
        <tr><td>${x} % ${y}</td><td>${x % y}</td></tr>
    </table>

    <h2>Relational Operators</h2>
    <table>
        <tr><th>Expression</th><th>Result</th></tr>
        <tr><td>${x} == ${y}</td><td>${x == y}</td></tr>
        <tr><td>${x} != ${y}</td><td>${x != y}</td></tr>
        <tr><td>${x} < ${y}</td><td>${x < y}</td></tr>
        <tr><td>${x} > ${y}</td><td>${x > y}</td></tr>
        <tr><td>${x} <= ${y}</td><td>${x <= y}</td></tr>
        <tr><td>${x} >= ${y}</td><td>${x >= y}</td></tr>
        <tr><td>${name} == "John"</td><td>${name == "John"}</td></tr>
    </table>

    <h2>Logical Operators</h2>
    <table>
        <tr><th>Expression</th><th>Result</th></tr>
        <tr><td>${x > 5 && y < 10}</td><td>${x > 5 && y < 10}</td></tr>
        <tr><td>${x > 5 || y > 10}</td><td>${x > 5 || y > 10}</td></tr>
        <tr><td>!(${x > y})</td><td>${!(x > y)}</td></tr>
    </table>

    <h2>Empty Operator</h2>
    <table>
```

```
        <tr><th>Expression</th><th>Result</th></tr>
        <tr><td>empty name</td><td>${empty name}</td></tr>
        <tr><td>empty emptyString</td><td>${empty emptyString}</td></tr>
        <tr><td>empty nullValue</td><td>${empty nullValue}</td></tr>
        <tr><td>empty emptyList</td><td>${empty emptyList}</td></tr>
        <tr><td>empty nonEmptyList</td><td>${empty nonEmptyList}</td></tr>
    </table>

    <h2>Conditional Operator</h2>
    <table>
        <tr><th>Expression</th><th>Result</th></tr>
        <tr><td>${x > y ? "x is greater" : "y is greater or equal"}</td><td>${x >
y ? "x is greater" : "y is greater or equal"}</td></tr>
        <tr><td>${empty name ? "Name is empty" : "Name is: ".concat(name)}</td>
<td>${empty name ? "Name is empty" : "Name is: ".concat(name)}</td></tr>
    </table>
</body>
</html>
```

EL Functions

EL functions extend EL's capabilities, allowing you to call static methods from Java classes. The JSTL library provides many useful functions:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>EL Functions Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>EL Functions Demonstration</h1>

    <%-- Set up some variables --%>
    <c:set var="text" value="Hello, World!" />
    <c:set var="longText" value="This is a longer text that we will use to
demonstrate various string functions in EL." />
    <c:set var="items" value="apple,banana,orange,grape,kiwi" />

    <h2>String Functions</h2>
    <table>
```

```
        <tr><th>Function</th><th>Result</th></tr>
        <tr><td>fn:length(text)</td><td>${fn:length(text)}</td></tr>
        <tr><td>fn:toUpperCase(text)</td><td>${fn:toUpperCase(text)}</td></tr>
        <tr><td>fn:toLowerCase(text)</td><td>${fn:toLowerCase(text)}</td></tr>
        <tr><td>fn:substring(text, 0, 5)</td><td>${fn:substring(text, 0, 5)}</td>
</tr>
        <tr><td>fn:substringAfter(text, ",")</td><td>${fn:substringAfter(text,
",")}</td></tr>
        <tr><td>fn:substringBefore(text, ",")</td><td>${fn:substringBefore(text,
",")}</td></tr>
        <tr><td>fn:replace(text, "Hello", "Hi")</td><td>${fn:replace(text,
"Hello", "Hi")}</td></tr>
        <tr><td>fn:indexOf(text, "World")</td><td>${fn:indexOf(text, "World")}
</td></tr>
        <tr><td>fn:contains(text, "Hello")</td><td>${fn:contains(text, "Hello")}
</td></tr>
        <tr><td>fn:containsIgnoreCase(text, "hello")</td>
<td>${fn:containsIgnoreCase(text, "hello")}</td></tr>
        <tr><td>fn:startsWith(text, "Hello")</td><td>${fn:startsWith(text,
"Hello")}</td></tr>
        <tr><td>fn:endsWith(text, "!")</td><td>${fn:endsWith(text, "!")}</td></tr>
        <tr><td>fn:trim(" Hello ")</td><td>[${fn:trim(" Hello ")}]</td></tr>
        <tr><td>fn:escapeXml("<h1>Title</h1>")</td><td>${fn:escapeXml("
<h1>Title</h1>")}</td></tr>
    </table>

    <h2>Collection Functions</h2>
    <p>String to split: ${items}</p>
    <p>Split into array:</p>
    <c:set var="itemsArray" value="${fn:split(items, ',')}" />
    <ul>
        <c:forEach var="item" items="${itemsArray}">
            <li>${item}</li>
        </c:forEach>
    </ul>
    <p>Join array with '-': ${fn:join(itemsArray, "-")}</p>
</body>
</html>
```

Creating Custom EL Functions

You can create your own EL functions by defining a Java class with static methods and declaring them in a TLD (Tag Library Descriptor) file:

1. Create a Java class with static methods:

```
package com.example.el;

public class CustomFunctions {

    // Calculate factorial
```

```
    public static long factorial(int n) {
        if (n <= 1) return 1;
        return n * factorial(n - 1);
    }

    // Check if a number is prime
    public static boolean isPrime(int n) {
        if (n <= 1) return false;
        if (n <= 3) return true;
        if (n % 2 == 0 || n % 3 == 0) return false;

        for (int i = 5; i * i <= n; i += 6) {
            if (n % i == 0 || n % (i + 2) == 0) {
                return false;
            }
        }
        return true;
    }

    // Format a number with commas
    public static String formatNumber(long number) {
        return String.format("%,d", number);
    }

    // Reverse a string
    public static String reverseString(String input) {
        return new StringBuilder(input).reverse().toString();
    }
}
```

2. Create a TLD file (WEB-INF/custom.tld):

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
        version="2.0">

    <tlib-version>1.0</tlib-version>
    <short-name>custom</short-name>
    <uri>http://example.com/custom-functions</uri>

    <function>
        <name>factorial</name>
        <function-class>com.example.el.CustomFunctions</function-class>
        <function-signature>long factorial(int)</function-signature>
    </function>

    <function>
        <name>isPrime</name>
        <function-class>com.example.el.CustomFunctions</function-class>
```

```
        <function-signature>boolean isPrime(int)</function-signature>
    </function>

    <function>
        <name>formatNumber</name>
        <function-class>com.example.el.CustomFunctions</function-class>
        <function-signature>java.lang.String formatNumber(long)</function-
signature>
    </function>

    <function>
        <name>reverseString</name>
        <function-class>com.example.el.CustomFunctions</function-class>
        <function-signature>java.lang.String reverseString(java.lang.String)
</function-signature>
    </function>
</taglib>
```

3. Use the custom functions in a JSP:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="custom" uri="http://example.com/custom-functions" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Custom EL Functions Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>Custom EL Functions Demonstration</h1>

    <h2>Mathematical Functions</h2>
    <table>
        <tr><th>Function</th><th>Result</th></tr>
        <tr><td>custom:factorial(5)</td><td>${custom:factorial(5)}</td></tr>
        <tr><td>custom:factorial(10)</td><td>${custom:factorial(10)}</td></tr>
        <tr><td>custom:isPrime(7)</td><td>${custom:isPrime(7)}</td></tr>
        <tr><td>custom:isPrime(10)</td><td>${custom:isPrime(10)}</td></tr>
        <tr><td>custom:formatNumber(1234567)</td>
<td>${custom:formatNumber(1234567)}</td></tr>
    </table>

    <h2>String Functions</h2>
    <table>
        <tr><th>Function</th><th>Result</th></tr>
        <tr><td>custom:reverseString("Hello World")</td>
```

```
<td>${custom:reverseString("Hello World")}</td></tr>
    </table>

    <h2>Interactive Example</h2>
    <form action="" method="get">
        <div>
            <label for="number">Enter a number:</label>
            <input type="number" id="number" name="number" value="${param.number}"
min="1" max="20">
            <button type="submit">Calculate</button>
        </div>
    </form>

    <c:if test="${not empty param.number}">
        <h3>Results for ${param.number}:</h3>
        <ul>
            <li>Factorial: ${custom:factorial(param.number)}</li>
            <li>Formatted: ${custom:formatNumber(custom:factorial(param.number))}
</li>
            <li>Is Prime: ${custom:isPrime(param.number)}</li>
        </ul>
    </c:if>
</body>
</html>
```

EL Best Practices

1. **Favor EL over Scriptlets**:
    - EL is more concise and readable
    - Separates Java code from presentation
2. **Use Dot Notation When Possible**:
    - Use `${user.name}` instead of `${user["name"]}`
    - Use bracket notation when property names have special characters or spaces
3. **Be Aware of Null Values**:
    - EL handles null values gracefully (displays nothing instead of "null")
    - Use empty operator to check for null or empty values
4. **Use JSTL with EL**:
    - Combine JSTL tags with EL for powerful functionality
    - JSTL provides tags for core functionality, formatting, SQL, XML processing
5. **Handle Missing Values**:
    - Use conditional operator for default values: `${empty user.name ? "Guest" : user.name}`
6. **Be Careful with Automatic Type Conversion**:

- EL automatically converts types, which can sometimes lead to unexpected results
- Be explicit when needed

7. **Secure Your EL Expressions**:
   - Use escapeXml function to prevent XSS attacks: `${fn:escapeXml(param.input)}`
8. **Create Custom Functions for Complex Logic**:
   - Move complex logic to custom EL functions
   - Keep the JSP clean and focused on presentation

3.5 JSP Standard Tag Library (JSTL)

The JSP Standard Tag Library (JSTL) provides a collection of useful JSP tags that encapsulate core functionality common to many JSP applications. JSTL helps eliminate the need for scriptlets, making JSPs cleaner and more maintainable.

JSTL Overview

JSTL consists of five tag libraries:

1. **Core Library**: Basic actions (variables, flow control, URL manipulation)
2. **Formatting Library**: Formatting actions (number, date, time formatting)
3. **SQL Library**: Database actions (querying databases, transactions)
4. **XML Library**: XML processing actions (parsing, transforming XML)
5. **Functions Library**: String manipulation functions

Setting Up JSTL

To use JSTL, you need to include the appropriate JAR files and taglib directives:

1. Add JSTL dependencies to your project:

For Maven:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

For Jakarta EE 9+ (Tomcat 10+):

```
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>2.0.0</version>
</dependency>
```

```
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>jakarta.servlet.jsp.jstl</artifactId>
    <version>2.0.0</version>
</dependency>
```

2. Add taglib directives to your JSP:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

For Jakarta EE 9+ (Tomcat 10+):

```
<%@ taglib prefix="c" uri="http://jakarta.ee/tags/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://jakarta.ee/tags/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://jakarta.ee/tags/jstl/sql" %>
<%@ taglib prefix="x" uri="http://jakarta.ee/tags/jstl/xml" %>
<%@ taglib prefix="fn" uri="http://jakarta.ee/tags/jstl/functions" %>
```

JSTL Core Tags

The Core library provides essential functionality for JSP pages:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSTL Core Tags Demo</title>
    <style>
        .even { background-color: #f2f2f2; }
        .odd { background-color: #ffffff; }
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; }
    </style>
</head>
<body>
    <h1>JSTL Core Tags Demonstration</h1>

    <%-- Variable Support --%>
    <h2>Variable Support</h2>
```

```
<%-- c:set - Sets a variable in a scope --%>
<c:set var="name" value="John Doe" />
<c:set var="age" value="30" />
<p>Name: ${name}</p>
<p>Age: ${age}</p>

<%-- Changing variable values --%>
<c:set var="age" value="${age + 5}" />
<p>Age after 5 years: ${age}</p>

<%-- Setting object properties --%>
<jsp:useBean id="user" class="com.example.User" scope="page" />
<c:set target="${user}" property="firstName" value="Jane" />
<c:set target="${user}" property="lastName" value="Smith" />
<p>User: ${user.firstName} ${user.lastName}</p>

<%-- c:remove - Removes a variable from a scope --%>
<c:remove var="age" />
<p>Age after remove: ${age} (should be empty)</p>

<%-- Flow Control --%>
<h2>Flow Control</h2>

<%-- c:if - Conditional execution --%>
<c:if test="${name == 'John Doe'}">
    <p>Hello, John!</p>
</c:if>

<%-- c:choose, c:when, c:otherwise - Multiple conditional execution --%>
<c:set var="score" value="${param.score}" />
<c:if test="${empty score}">
    <c:set var="score" value="75" />
</c:if>

<p>Score: ${score}</p>
<c:choose>
    <c:when test="${score >= 90}">
        <p>Grade: A</p>
    </c:when>
    <c:when test="${score >= 80}">
        <p>Grade: B</p>
    </c:when>
    <c:when test="${score >= 70}">
        <p>Grade: C</p>
    </c:when>
    <c:when test="${score >= 60}">
        <p>Grade: D</p>
    </c:when>
    <c:otherwise>
        <p>Grade: F</p>
    </c:otherwise>
</c:choose>

<%-- c:forEach - Iteration --%>
```

```
    <h2>Iteration</h2>

    <%-- Iterating over a range --%>
    <h3>Numbers from 1 to 5:</h3>
    <ul>
        <c:forEach var="i" begin="1" end="5">
            <li>Number ${i}</li>
        </c:forEach>
    </ul>

    <%-- Iterating over a collection --%>
    <c:set var="fruits" value="${['Apple', 'Banana', 'Orange', 'Mango',
'Grapes']}" />

    <h3>Fruits:</h3>
    <table>
        <tr>
            <th>#</th>
            <th>Name</th>
            <th>Status</th>
        </tr>
        <c:forEach var="fruit" items="${fruits}" varStatus="status">
            <tr class="${status.count % 2 == 0 ? 'even' : 'odd'}">
                <td>${status.count}</td>
                <td>${fruit}</td>
                <td>
                    <c:choose>
                        <c:when test="${status.first}">First item</c:when>
                        <c:when test="${status.last}">Last item</c:when>
                        <c:otherwise>Middle item</c:otherwise>
                    </c:choose>
                </td>
            </tr>
        </c:forEach>
    </table>

    <%-- c:forTokens - Iterating over tokens --%>
    <h3>Comma-separated Values:</h3>
    <c:set var="csvData" value="apple,banana,orange,grape,kiwi" />
    <ul>
        <c:forTokens var="fruit" items="${csvData}" delims=",">
            <li>${fruit}</li>
        </c:forTokens>
    </ul>

    <%-- URL Actions --%>
    <h2>URL Actions</h2>

    <%-- c:url - URL formatting --%>
    <c:url var="myUrl" value="/products">
        <c:param name="category" value="electronics" />
        <c:param name="sort" value="price" />
        <c:param name="order" value="asc" />
    </c:url>
```

```
    <p>Formatted URL: ${myUrl}</p>

    <%-- c:redirect - Redirect to another URL (commented out) --%>
    <%-- <c:redirect url="${myUrl}" /> --%>

    <%-- c:import - Imports content from another URL --%>
    <h3>Imported Content:</h3>
    <div style="border: 1px solid #ddd; padding: 10px; margin: 10px 0;">
        <c:catch var="importError">
            <c:import url="/WEB-INF/fragments/sample-content.jsp" />
        </c:catch>
        <c:if test="${not empty importError}">
            <p>Error importing content: ${importError.message}</p>
        </c:if>
    </div>

    <%-- Other Tags --%>
    <h2>Other Tags</h2>

    <%-- c:catch - Catch exceptions --%>
    <c:catch var="divideError">
        <% int result = 10 / 0; %>
        <p>Result: <%= result %></p>
    </c:catch>
    <c:if test="${not empty divideError}">
        <p>An error occurred: ${divideError.message}</p>
    </c:if>

    <%-- c:out - Outputs expressions with escaping --%>
    <h3>Output with Escaping:</h3>
    <c:set var="htmlContent" value="<strong>This is bold</strong>" />
    <p>With c:out (escaped): <c:out value="${htmlContent}" /></p>
    <p>Without c:out (not escaped): ${htmlContent}</p>

    <h3>Try It Out:</h3>
    <form action="" method="get">
        <div>
            <label for="score">Enter a score (0-100):</label>
            <input type="number" id="score" name="score" value="${param.score}"
min="0" max="100">
            <button type="submit">Get Grade</button>
        </div>
    </form>
</body>
</html>
```

Sample content file (WEB-INF/fragments/sample-content.jsp):

```
<div>
    <h4>Sample Imported Content</h4>
    <p>This content is being imported from another JSP file.</p>
```

```
    <p>Current time: <%= new java.util.Date() %></p>
</div>
```

JSTL Formatting Tags

The formatting library provides tags for formatting and parsing numbers, dates, and messages:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSTL Formatting Tags Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>JSTL Formatting Tags Demonstration</h1>

    <%-- Setup variables --%>
    <c:set var="now" value="<%= new java.util.Date() %>" />
    <c:set var="amount" value="12345.67" />
    <c:set var="temperature" value="36.5" />

    <%-- Number Formatting --%>
    <h2>Number Formatting</h2>
    <table>
        <tr><th>Format</th><th>Result</th></tr>

        <%-- Basic number formatting --%>
        <tr>
            <td>Default Number</td>
            <td><fmt:formatNumber value="${amount}" /></td>
        </tr>

        <%-- Currency formatting --%>
        <tr>
            <td>Currency (default locale)</td>
            <td><fmt:formatNumber value="${amount}" type="currency" /></td>
        </tr>

        <tr>
            <td>Currency (US)</td>
            <td>
                <fmt:setLocale value="en_US" />
```

```
                <fmt:formatNumber value="${amount}" type="currency" />
            </td>
        </tr>

        <tr>
            <td>Currency (UK)</td>
            <td>
                <fmt:setLocale value="en_GB" />
                <fmt:formatNumber value="${amount}" type="currency" />
            </td>
        </tr>

        <tr>
            <td>Currency (Japan)</td>
            <td>
                <fmt:setLocale value="ja_JP" />
                <fmt:formatNumber value="${amount}" type="currency" />
            </td>
        </tr>

        <tr>
            <td>Currency (custom pattern)</td>
            <td>
                <fmt:setLocale value="en_US" />
                <fmt:formatNumber value="${amount}" type="currency"
currencySymbol="$" />
            </td>
        </tr>

        <%-- Percentage formatting --%>
        <tr>
            <td>Percentage</td>
            <td><fmt:formatNumber value="0.25" type="percent" /></td>
        </tr>

        <tr>
            <td>Percentage (custom)</td>
            <td><fmt:formatNumber value="0.25" type="percent"
maxFractionDigits="2" /></td>
        </tr>

        <%-- Custom patterns --%>
        <tr>
            <td>Custom Pattern (#,##0.00)</td>
            <td><fmt:formatNumber value="${amount}" pattern="#,##0.00" /></td>
        </tr>

        <tr>
            <td>Temperature (0.0°C)</td>
            <td><fmt:formatNumber value="${temperature}" pattern="0.0"/>°C</td>
        </tr>
    </table>

    <%-- Date Formatting --%>
```

```
    <h2>Date Formatting</h2>
    <table>
        <tr><th>Format</th><th>Result</th></tr>

        <%-- Basic date formatting --%>
        <tr>
            <td>Default Date</td>
            <td><fmt:formatDate value="${now}" /></td>
        </tr>

        <tr>
            <td>Default Time</td>
            <td><fmt:formatDate value="${now}" type="time" /></td>
        </tr>

        <tr>
            <td>Default Date and Time</td>
            <td><fmt:formatDate value="${now}" type="both" /></td>
        </tr>

        <%-- Date styles --%>
        <tr>
            <td>Short Date</td>
            <td><fmt:formatDate value="${now}" type="date" dateStyle="short" />
</td>
        </tr>

        <tr>
            <td>Medium Date</td>
            <td><fmt:formatDate value="${now}" type="date" dateStyle="medium" />
</td>
        </tr>

        <tr>
            <td>Long Date</td>
            <td><fmt:formatDate value="${now}" type="date" dateStyle="long" />
</td>
        </tr>

        <tr>
            <td>Full Date</td>
            <td><fmt:formatDate value="${now}" type="date" dateStyle="full" />
</td>
        </tr>

        <%-- Time styles --%>
        <tr>
            <td>Short Time</td>
            <td><fmt:formatDate value="${now}" type="time" timeStyle="short" />
</td>
        </tr>

        <tr>
            <td>Medium Time</td>
```

```
            <td><fmt:formatDate value="${now}" type="time" timeStyle="medium" />
</td>
        </tr>

        <tr>
            <td>Long Time</td>
            <td><fmt:formatDate value="${now}" type="time" timeStyle="long" />
</td>
        </tr>

        <tr>
            <td>Full Time</td>
            <td><fmt:formatDate value="${now}" type="time" timeStyle="full" />
</td>
        </tr>

        <%-- Combined styles --%>
        <tr>
            <td>Full Date and Time</td>
            <td><fmt:formatDate value="${now}" type="both" dateStyle="full"
timeStyle="full" /></td>
        </tr>

        <%-- Custom patterns --%>
        <tr>
            <td>Custom Pattern (yyyy-MM-dd HH:mm:ss)</td>
            <td><fmt:formatDate value="${now}" pattern="yyyy-MM-dd HH:mm:ss" />
</td>
        </tr>

        <tr>
            <td>ISO-8601 Pattern</td>
            <td><fmt:formatDate value="${now}" pattern="yyyy-MM-
dd'T'HH:mm:ss.SSSXXX" /></td>
        </tr>
    </table>

    <%-- Parsing Examples --%>
    <h2>Parsing Examples</h2>

    <%-- Parse a date string --%>
    <c:set var="dateString" value="2023-01-15" />
    <fmt:parseDate var="parsedDate" value="${dateString}" pattern="yyyy-MM-dd" />
    <p>Parsed Date: <fmt:formatDate value="${parsedDate}" type="both"
dateStyle="full" timeStyle="medium" /></p>

    <%-- Parse a number string --%>
    <c:set var="numberString" value="1,234.56" />
    <fmt:parseNumber var="parsedNumber" value="${numberString}" />
    <p>Parsed Number: ${parsedNumber}</p>

    <%-- Timezone Examples --%>
    <h2>Timezone Examples</h2>
    <table>
```

```
        <tr><th>Timezone</th><th>Date and Time</th></tr>

        <tr>
            <td>Default</td>
            <td><fmt:formatDate value="${now}" type="both" dateStyle="full"
timeStyle="full" /></td>
        </tr>

        <tr>
            <td>GMT</td>
            <td>
                <fmt:timeZone value="GMT">
                    <fmt:formatDate value="${now}" type="both" dateStyle="full"
timeStyle="full" />
                </fmt:timeZone>
            </td>
        </tr>

        <tr>
            <td>America/New_York</td>
            <td>
                <fmt:timeZone value="America/New_York">
                    <fmt:formatDate value="${now}" type="both" dateStyle="full"
timeStyle="full" />
                </fmt:timeZone>
            </td>
        </tr>

        <tr>
            <td>Asia/Tokyo</td>
            <td>
                <fmt:timeZone value="Asia/Tokyo">
                    <fmt:formatDate value="${now}" type="both" dateStyle="full"
timeStyle="full" />
                </fmt:timeZone>
            </td>
        </tr>
    </table>

    <%-- Internationalization Examples --%>
    <h2>Internationalization Examples</h2>

    <%-- Resource Bundle Message --%>
    <p>
        <fmt:setLocale value="en_US" />
        <fmt:setBundle basename="com.example.messages" />

        <fmt:message key="greeting" var="greeting" />
        English Greeting: ${greeting}
    </p>

    <p>
        <fmt:setLocale value="fr_FR" />
        <fmt:setBundle basename="com.example.messages" />
```

```
        <fmt:message key="greeting" var="greeting" />
        French Greeting: ${greeting}
    </p>

    <p>
        <fmt:setLocale value="es_ES" />
        <fmt:setBundle basename="com.example.messages" />

        <fmt:message key="greeting" var="greeting" />
        Spanish Greeting: ${greeting}
    </p>
</body>
</html>
```

Resource Bundle file (com/example/messages_en_US.properties):

```
greeting=Hello, welcome to our application!
```

Resource Bundle file (com/example/messages_fr_FR.properties):

```
greeting=Bonjour, bienvenue à notre application!
```

Resource Bundle file (com/example/messages_es_ES.properties):

```
greeting=¡Hola, bienvenido a nuestra aplicación!
```

JSTL SQL Tags

The SQL library provides tags for interacting with databases:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSTL SQL Tags Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; margin-bottom: 20px; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
```

```
        form { margin-bottom: 20px; padding: 15px; background-color: #f8f9fa;
border-radius: 5px; }
        .form-group { margin-bottom: 10px; }
        label { display: inline-block; width: 100px; }
    </style>
</head>
<body>
    <h1>JSTL SQL Tags Demonstration</h1>

    <%-- Set up database connection --%>
    <sql:setDataSource var="dataSource" driver="com.mysql.cj.jdbc.Driver"
                       url="jdbc:mysql://localhost:3306/jspdb"
                       user="jspuser" password="jsppassword" />

    <%-- Create table if not exists (executed only once for demo) --%>
    <c:if test="${param.createTable eq 'true'}">
        <sql:update dataSource="${dataSource}" var="result">
            CREATE TABLE IF NOT EXISTS employees (
                id INT AUTO_INCREMENT PRIMARY KEY,
                name VARCHAR(100) NOT NULL,
                email VARCHAR(100) NOT NULL,
                department VARCHAR(50) NOT NULL,
                salary DECIMAL(10,2) NOT NULL,
                hire_date DATE NOT NULL
            )
        </sql:update>
        <p>Table created or already exists.</p>
    </c:if>

    <%-- Insert demo data if requested --%>
    <c:if test="${param.insertDemo eq 'true'}">
        <sql:update dataSource="${dataSource}" var="result">
            INSERT INTO employees (name, email, department, salary, hire_date)
VALUES
            ('John Smith', 'john@example.com', 'IT', 75000.00, '2020-01-15'),
            ('Mary Johnson', 'mary@example.com', 'HR', 65000.00, '2019-06-20'),
            ('Robert Brown', 'robert@example.com', 'Finance', 85000.00, '2021-03-
10'),
            ('Jennifer Davis', 'jennifer@example.com', 'Marketing', 70000.00,
'2018-11-05'),
            ('Michael Wilson', 'michael@example.com', 'IT', 78000.00, '2020-09-
30')
        </sql:update>
        <p>Demo data inserted. ${result} rows affected.</p>
    </c:if>

    <%-- Insert a new employee if form is submitted --%>
    <c:if test="${not empty param.name and not empty param.email and not empty
param.department and not empty param.salary and not empty param.hireDate}">
        <sql:update dataSource="${dataSource}" var="result">
            INSERT INTO employees (name, email, department, salary, hire_date)
VALUES (?, ?, ?, ?, ?)
            <sql:param value="${param.name}" />
            <sql:param value="${param.email}" />
```

```
            <sql:param value="${param.department}" />
            <sql:param value="${param.salary}" />
            <sql:param value="${param.hireDate}" />
        </sql:update>
        <p>Employee added successfully. ${result} row(s) affected.</p>
    </c:if>

    <%-- Delete an employee if requested --%>
    <c:if test="${not empty param.deleteId}">
        <sql:update dataSource="${dataSource}" var="result">
            DELETE FROM employees WHERE id = ?
            <sql:param value="${param.deleteId}" />
        </sql:update>
        <p>Employee deleted. ${result} row(s) affected.</p>
    </c:if>

    <%-- Query all employees --%>
    <sql:query dataSource="${dataSource}" var="employees">
        SELECT * FROM employees
        <c:if test="${not empty param.department}">
            WHERE department = ?
            <sql:param value="${param.department}" />
        </c:if>
        ORDER BY id
    </sql:query>

    <%-- Display employees table --%>
    <h2>Employees List</h2>

    <%-- Filter by department --%>
    <form action="" method="get">
        <div class="form-group">
            <label for="department">Department:</label>
            <select id="department" name="department">
                <option value="">All Departments</option>
                <option value="IT" ${param.department eq 'IT' ? 'selected' :
''}>IT</option>
                <option value="HR" ${param.department eq 'HR' ? 'selected' :
''}>HR</option>
                <option value="Finance" ${param.department eq 'Finance' ?
'selected' : ''}>Finance</option>
                <option value="Marketing" ${param.department eq 'Marketing' ?
'selected' : ''}>Marketing</option>
            </select>
            <button type="submit">Filter</button>
        </div>
    </form>

    <table>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
            <th>Department</th>
```

```
            <th>Salary</th>
            <th>Hire Date</th>
            <th>Actions</th>
        </tr>
        <c:forEach var="employee" items="${employees.rows}">
            <tr>
                <td>${employee.id}</td>
                <td>${employee.name}</td>
                <td>${employee.email}</td>
                <td>${employee.department}</td>
                <td><fmt:formatNumber value="${employee.salary}" type="currency"
/></td>
                <td><fmt:formatDate value="${employee.hire_date}" pattern="yyyy-
MM-dd" /></td>
                <td><a href="?deleteId=${employee.id}" onclick="return
confirm('Are you sure?')">Delete</a></td>
            </tr>
        </c:forEach>
    </table>

    <%-- Get department summary --%>
    <sql:query dataSource="${dataSource}" var="deptSummary">
        SELECT department, COUNT(*) as count, AVG(salary) as avg_salary,
MAX(salary) as max_salary, MIN(salary) as min_salary
        FROM employees
        GROUP BY department
        ORDER BY department
    </sql:query>

    <%-- Display department summary --%>
    <h2>Department Summary</h2>
    <table>
        <tr>
            <th>Department</th>
            <th>Employee Count</th>
            <th>Average Salary</th>
            <th>Maximum Salary</th>
            <th>Minimum Salary</th>
        </tr>
        <c:forEach var="dept" items="${deptSummary.rows}">
            <tr>
                <td>${dept.department}</td>
                <td>${dept.count}</td>
                <td><fmt:formatNumber value="${dept.avg_salary}" type="currency"
/></td>
                <td><fmt:formatNumber value="${dept.max_salary}" type="currency"
/></td>
                <td><fmt:formatNumber value="${dept.min_salary}" type="currency"
/></td>
            </tr>
        </c:forEach>
    </table>

    <%-- Transaction example --%>
```

```
    <h2>Transaction Example</h2>
    <c:if test="${param.runTransaction eq 'true'}">
        <sql:transaction dataSource="${dataSource}">
            <sql:update var="step1">
                UPDATE employees SET salary = salary * 1.05 WHERE department =
'IT'
            </sql:update>
            <sql:update var="step2">
                UPDATE employees SET salary = salary * 1.03 WHERE department !=
'IT'
            </sql:update>
            <p>Transaction completed. IT employees got 5% raise, others got 3%
raise.</p>
        </sql:transaction>
    </c:if>
    <p><a href="?runTransaction=true">Run Salary Increase Transaction</a></p>

    <%-- Add employee form --%>
    <h2>Add New Employee</h2>
    <form action="" method="post">
        <div class="form-group">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>
        </div>
        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
        </div>
        <div class="form-group">
            <label for="department">Department:</label>
            <select id="department" name="department" required>
                <option value="">Select Department</option>
                <option value="IT">IT</option>
                <option value="HR">HR</option>
                <option value="Finance">Finance</option>
                <option value="Marketing">Marketing</option>
            </select>
        </div>
        <div class="form-group">
            <label for="salary">Salary:</label>
            <input type="number" id="salary" name="salary" step="0.01" min="0"
required>
        </div>
        <div class="form-group">
            <label for="hireDate">Hire Date:</label>
            <input type="date" id="hireDate" name="hireDate" required>
        </div>
        <div class="form-group">
            <button type="submit">Add Employee</button>
        </div>
    </form>

    <%-- Setup Links --%>
    <h2>Setup</h2>
```

```
    <p><a href="?createTable=true">Create Table</a> | <a href="?
insertDemo=true">Insert Demo Data</a></p>
</body>
</html>
```

JSTL XML Tags

The XML library provides tags for parsing and processing XML:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSTL XML Tags Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; margin-bottom: 20px; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
        pre { background-color: #f5f5f5; padding: 10px; border-radius: 5px;
overflow-x: auto; }
        .highlight { background-color: #ffffcc; }
    </style>
</head>
<body>
    <h1>JSTL XML Tags Demonstration</h1>

    <%-- Define an XML document --%>
    <c:set var="bookXml">
        <books>
            <book id="1">
                <title>Harry Potter and the Philosopher's Stone</title>
                <author>J.K. Rowling</author>
                <genre>Fantasy</genre>
                <price>19.99</price>
                <publish_date>1997-06-26</publish_date>
            </book>
            <book id="2">
                <title>The Hobbit</title>
                <author>J.R.R. Tolkien</author>
                <genre>Fantasy</genre>
                <price>15.99</price>
                <publish_date>1937-09-21</publish_date>
            </book>
            <book id="3">
                <title>To Kill a Mockingbird</title>
                <author>Harper Lee</author>
                <genre>Fiction</genre>
```

```
                <price>12.99</price>
                <publish_date>1960-07-11</publish_date>
            </book>
            <book id="4">
                <title>1984</title>
                <author>George Orwell</author>
                <genre>Dystopian</genre>
                <price>10.99</price>
                <publish_date>1949-06-08</publish_date>
            </book>
            <book id="5">
                <title>The Great Gatsby</title>
                <author>F. Scott Fitzgerald</author>
                <genre>Fiction</genre>
                <price>11.99</price>
                <publish_date>1925-04-10</publish_date>
            </book>
        </books>
    </c:set>

    <%-- Parse the XML document --%>
    <x:parse xml="${bookXml}" var="bookDoc" />

    <h2>XML Document</h2>
    <pre>${bookXml}</pre>

    <h2>XML Parsing and Accessing</h2>

    <%-- Access a specific element --%>
    <h3>First Book Title:</h3>
    <p><x:out select="$bookDoc/books/book[1]/title" /></p>

    <h3>All Book Titles:</h3>
    <ul>
        <x:forEach select="$bookDoc/books/book" var="book">
            <li><x:out select="$book/title" /></li>
        </x:forEach>
    </ul>

    <h2>XML Conditional Processing</h2>

    <%-- Conditional logic with XML --%>
    <h3>Books by Genre:</h3>
    <table>
        <tr>
            <th>Title</th>
            <th>Author</th>
            <th>Genre</th>
            <th>Price</th>
        </tr>
        <x:forEach select="$bookDoc/books/book" var="book">
            <tr>
                <td><x:out select="$book/title" /></td>
                <td><x:out select="$book/author" /></td>
```

```
                <td>
                    <x:out select="$book/genre" />
                    <x:if select="$book/genre = 'Fantasy'">
                        <span class="highlight"> (Fantasy Book)</span>
                    </x:if>
                </td>
                <td>
                    $<x:out select="$book/price" />
                    <x:choose>
                        <x:when select="$book/price > 15">
                            <span style="color: red;"> (Expensive)</span>
                        </x:when>
                        <x:when select="$book/price > 10">
                            <span style="color: orange;"> (Moderate)</span>
                        </x:when>
                        <x:otherwise>
                            <span style="color: green;"> (Affordable)</span>
                        </x:otherwise>
                    </x:choose>
                </td>
            </tr>
        </x:forEach>
    </table>

    <h2>XML Transformation with XSLT</h2>

    <%-- Define an XSLT stylesheet --%>
    <c:set var="xsltString">
        <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
            <xsl:output method="html" indent="yes"/>

            <xsl:template match="/">
                <div class="book-catalog">
                    <h2>Book Catalog</h2>
                    <table border="1">
                        <tr style="background-color: #4CAF50; color: white;">
                            <th>ID</th>
                            <th>Title</th>
                            <th>Author</th>
                            <th>Genre</th>
                            <th>Price</th>
                            <th>Published</th>
                        </tr>
                        <xsl:for-each select="books/book">
                            <xsl:sort select="publish_date" />
                            <tr>
                                <td><xsl:value-of select="@id"/></td>
                                <td><xsl:value-of select="title"/></td>
                                <td><xsl:value-of select="author"/></td>
                                <td><xsl:value-of select="genre"/></td>
                                <td>
                                    <xsl:choose>
                                        <xsl:when test="price > 15">
```

```
                                            <span style="color: red">$<xsl:value-
of select="price"/></span>
                                        </xsl:when>
                                        <xsl:otherwise>
                                            <span style="color:
green">$<xsl:value-of select="price"/></span>
                                        </xsl:otherwise>
                                    </xsl:choose>
                                </td>
                                <td><xsl:value-of select="publish_date"/></td>
                            </tr>
                        </xsl:for-each>
                    </table>
                </div>
            </xsl:template>
        </xsl:stylesheet>
    </c:set>

    <%-- Parse the XSLT stylesheet --%>
    <x:parse xml="${xsltString}" var="xsltDoc" />

    <%-- Transform the XML using the XSLT stylesheet --%>
    <x:transform xml="${bookXml}" xslt="${xsltDoc}" />

    <h2>XML Filtering with XPath</h2>

    <h3>Fantasy Books Only:</h3>
    <ul>
        <x:forEach select="$bookDoc/books/book[genre='Fantasy']">
            <li><x:out select="title" /> by <x:out select="author" /></li>
        </x:forEach>
    </ul>

    <h3>Books Published After 1950:</h3>
    <ul>
        <x:forEach select="$bookDoc/books/book[publish_date > '1950-01-01']">
            <li>
                <x:out select="title" />
                (<x:out select="publish_date" />)
            </li>
        </x:forEach>
    </ul>

    <h3>Books Cheaper Than $15:</h3>
    <ul>
        <x:forEach select="$bookDoc/books/book[price < 15]">
            <li>
                <x:out select="title" /> - $<x:out select="price" />
            </li>
        </x:forEach>
    </ul>

    <h2>More Complex XPath Queries</h2>
```

```
    <h3>Books by Genre Count:</h3>
    <x:forEach select="$bookDoc/books/book/genre[not(. = preceding::genre)]"
var="genre">
        <p>
            <x:out select="$genre" />:
            <x:out select="count($bookDoc/books/book[genre = $genre])" /> book(s)
        </p>
    </x:forEach>

    <h3>Average Book Price:</h3>
    <x:set var="total" select="sum($bookDoc/books/book/price)" />
    <x:set var="count" select="count($bookDoc/books/book)" />
    <p>Average Price: $<x:out select="$total div $count" /></p>
</body>
</html>
```

JSTL Functions

The Functions library provides a collection of functions for string manipulation:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSTL Functions Demo</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>JSTL Functions Demonstration</h1>

    <%-- Set up test strings --%>
    <c:set var="text1" value="Hello, World!" />
    <c:set var="text2" value="   This is a trimmed string.   " />
    <c:set var="text3" value="<p>This contains <strong>HTML</strong> tags</p>" />
    <c:set var="items" value="apple,banana,orange,grape,kiwi" />
    <c:set var="longText" value="This is a longer text that shows how we can
demonstrate various string functions in JSTL." />

    <h2>String Length Functions</h2>
    <table>
        <tr><th>Function</th><th>Result</th></tr>
        <tr>
            <td>fn:length(text1)</td>
```

```
            <td>${fn:length(text1)}</td>
        </tr>
        <tr>
            <td>fn:length(text2)</td>
            <td>${fn:length(text2)}</td>
        </tr>
        <tr>
            <td>fn:length(text3)</td>
            <td>${fn:length(text3)}</td>
        </tr>
        <tr>
            <td>fn:length(items.split(","))</td>
            <td>${fn:length(fn:split(items, ","))}</td>
        </tr>
    </table>

    <h2>String Case Functions</h2>
    <table>
        <tr><th>Function</th><th>Result</th></tr>
        <tr>
            <td>fn:toUpperCase(text1)</td>
            <td>${fn:toUpperCase(text1)}</td>
        </tr>
        <tr>
            <td>fn:toLowerCase(text1)</td>
            <td>${fn:toLowerCase(text1)}</td>
        </tr>
    </table>

    <h2>String Manipulation Functions</h2>
    <table>
        <tr><th>Function</th><th>Result</th></tr>
        <tr>
            <td>fn:substring(text1, 0, 5)</td>
            <td>${fn:substring(text1, 0, 5)}</td>
        </tr>
        <tr>
            <td>fn:substringAfter(text1, ",")</td>
            <td>${fn:substringAfter(text1, ",")}</td>
        </tr>
        <tr>
            <td>fn:substringBefore(text1, ",")</td>
            <td>${fn:substringBefore(text1, ",")}</td>
        </tr>
        <tr>
            <td>fn:replace(text1, "Hello", "Hi")</td>
            <td>${fn:replace(text1, "Hello", "Hi")}</td>
        </tr>
        <tr>
            <td>fn:trim(text2)</td>
            <td>"${fn:trim(text2)}"</td>
        </tr>
    </table>
```

```
    <h2>String Testing Functions</h2>
    <table>
        <tr><th>Function</th><th>Result</th></tr>
        <tr>
            <td>fn:contains(text1, "World")</td>
            <td>${fn:contains(text1, "World")}</td>
        </tr>
        <tr>
            <td>fn:contains(text1, "Java")</td>
            <td>${fn:contains(text1, "Java")}</td>
        </tr>
        <tr>
            <td>fn:containsIgnoreCase(text1, "world")</td>
            <td>${fn:containsIgnoreCase(text1, "world")}</td>
        </tr>
        <tr>
            <td>fn:startsWith(text1, "Hello")</td>
            <td>${fn:startsWith(text1, "Hello")}</td>
        </tr>
        <tr>
            <td>fn:endsWith(text1, "!")</td>
            <td>${fn:endsWith(text1, "!")}</td>
        </tr>
        <tr>
            <td>fn:indexOf(text1, "o")</td>
            <td>${fn:indexOf(text1, "o")}</td>
        </tr>
    </table>

    <h2>Array/Collection Functions</h2>
    <c:set var="itemsArray" value="${fn:split(items, ',')}" />

    <h3>Split String to Array:</h3>
    <ul>
        <c:forEach var="item" items="${itemsArray}">
            <li>${item}</li>
        </c:forEach>
    </ul>

    <h3>Join Array to String:</h3>
    <p>Original: ${items}</p>
    <p>Joined with '-': ${fn:join(itemsArray, "-")}</p>

    <h2>HTML/XML Escaping</h2>
    <table>
        <tr><th>Function</th><th>Result</th></tr>
        <tr>
            <td>Original HTML</td>
            <td>${text3}</td>
        </tr>
        <tr>
            <td>fn:escapeXml(text3)</td>
            <td>${fn:escapeXml(text3)}</td>
        </tr>
```

```
    </table>

    <h2>Function Usage in Conditions</h2>
    <c:forEach var="item" items="${itemsArray}">
        <c:if test="${fn:length(item) > 5}">
            <p>${item} has more than 5 characters</p>
        </c:if>
    </c:forEach>

    <h2>Combining Functions</h2>
    <p>First 20 characters of longText: ${fn:substring(longText, 0, 20)}...</p>
    <p>longText in uppercase: ${fn:toUpperCase(longText)}</p>
    <p>Number of words in longText: ${fn:length(fn:split(longText, " "))}</p>

    <h2>Try It Yourself</h2>
    <form action="" method="get">
        <div>
            <label for="testString">Enter a string:</label>
            <input type="text" id="testString" name="testString"
value="${param.testString}" size="50">
            <button type="submit">Analyze</button>
        </div>
    </form>

    <c:if test="${not empty param.testString}">
        <h3>Analysis Results for: ${param.testString}</h3>
        <table>
            <tr><th>Function</th><th>Result</th></tr>
            <tr>
                <td>Length</td>
                <td>${fn:length(param.testString)}</td>
            </tr>
            <tr>
                <td>Uppercase</td>
                <td>${fn:toUpperCase(param.testString)}</td>
            </tr>
            <tr>
                <td>Lowercase</td>
                <td>${fn:toLowerCase(param.testString)}</td>
            </tr>
            <tr>
                <td>First 5 chars (if available)</td>
                <td>
                    <c:if test="${fn:length(param.testString) >= 5}">
                        ${fn:substring(param.testString, 0, 5)}
                    </c:if>
                    <c:if test="${fn:length(param.testString) < 5}">
                        (string too short)
                    </c:if>
                </td>
            </tr>
            <tr>
                <td>Contains 'a'</td>
                <td>${fn:contains(param.testString, 'a')}</td>
```

```
            </tr>
            <tr>
                <td>Word count</td>
                <td>${fn:length(fn:split(param.testString, ' '))}</td>
            </tr>
        </table>
    </c:if>
</body>
</html>
```

JSTL Best Practices

1. **Choose the Right Tags**:
    - Use core tags for control flow and variable manipulation
    - Use formatting tags for localization and number/date formatting
    - Use functions for string manipulation
    - Use SQL tags sparingly and only for simple database operations
    - Consider custom tags for complex business logic
2. **Minimize Scriptlets**:
    - Replace scriptlets with JSTL tags and EL
    - Move complex logic to beans or custom tags
3. **Optimize Performance**:
    - Cache results of expensive operations
    - Use conditional evaluation (`<c:if>`, `<c:choose>`) to avoid unnecessary processing
    - Be mindful of JSTL sql:query performance in high-traffic applications
4. **Secure JSTL Usage**:
    - Always use `<c:out>` or `fn:escapeXml()` for displaying user input
    - Validate input parameters before using them in SQL queries
    - Use `<sql:param>` to prevent SQL injection
5. **Maintainability Tips**:
    - Keep JSP pages focused on presentation
    - Use consistent naming conventions for variables
    - Comment complex JSTL expressions
    - Break large pages into smaller, reusable components
6. **Error Handling**:
    - Use `<c:catch>` to handle exceptions gracefully
    - Provide user-friendly error messages
    - Log errors for debugging
7. **Internationalization**:

- Use `<fmt:message>` for localizable text
- Store messages in resource bundles
- Use `<fmt:formatDate>` and `<fmt:formatNumber>` for locale-aware formatting

3.6 Custom Tag Development

Custom JSP tags allow you to encapsulate complex functionality in reusable components. This improves code organization, maintainability, and creates a cleaner separation between presentation and logic.

Why Create Custom Tags?

1. **Reusability**: Package functionality for use across multiple JSPs
2. **Maintainability**: Centralize complex logic in one place
3. **Encapsulation**: Hide implementation details
4. **Separation of concerns**: Keep JSPs focused on presentation
5. **Domain-specific language**: Create tags that reflect your application's domain

Types of Custom Tags

There are several approaches to creating custom tags:

1. **Simple Tag Handlers**: Using the SimpleTag interface (JSP 2.0+)
2. **Classic Tag Handlers**: Using the Tag interface (older approach)
3. **Tag Files**: JSP fragments that act as custom tags
4. **Functions**: Custom EL functions

Creating a Simple Tag Handler

Simple tag handlers implement the `javax.servlet.jsp.tagext.SimpleTag` interface or extend `javax.servlet.jsp.tagext.SimpleTagSupport`.

1. Create the Tag Handler Class:

```
package com.example.tags;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class GreetingTag extends SimpleTagSupport {

    private String name;
    private String color;

    // Setters for attributes
    public void setName(String name) {
        this.name = name;
    }

    public void setColor(String color) {
```

```
        this.color = color;
    }

    @Override
    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();

        if (name == null || name.trim().length() == 0) {
            name = "Guest";
        }

        if (color == null || color.trim().length() == 0) {
            color = "black";
        }

        out.println("<div style=\"color: " + color + ";\">");
        out.println("    <h2>Hello, " + name + "!</h2>");
        out.println("    <p>Welcome to our website.</p>");
        out.println("</div>");
    }
}
```

2. Define the Tag Library Descriptor (TLD):

Create a file named `custom.tld` in the WEB-INF directory:

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
        version="2.0">

    <tlib-version>1.0</tlib-version>
    <short-name>custom</short-name>
    <uri>http://example.com/custom-tags</uri>

    <tag>
        <n>greeting</n>
        <tag-class>com.example.tags.GreetingTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <n>name</n>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <n>color</n>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
```

```
    </tag>
</taglib>
```

3. Use the Custom Tag in a JSP:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="my" uri="http://example.com/custom-tags" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Custom Tag Demo</title>
</head>
<body>
    <h1>Custom Tag Demonstration</h1>

    <my:greeting name="John" color="blue" />

    <my:greeting name="Mary" color="green" />

    <my:greeting name="${param.name}" color="red" />

    <my:greeting />

    <h2>Try It</h2>
    <form action="" method="get">
        <div>
            <label for="name">Your Name:</label>
            <input type="text" id="name" name="name">
            <button type="submit">Submit</button>
        </div>
    </form>
</body>
</html>
```

Creating Tags with Body Content

You can create tags that process their body content:

1. Create the Tag Handler:

```
package com.example.tags;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HighlightTag extends SimpleTagSupport {
```

```
    private String color;
    private boolean bold;

    public void setColor(String color) {
        this.color = color;
    }

    public void setBold(boolean bold) {
        this.bold = bold;
    }

    @Override
    public void doTag() throws JspException, IOException {
        StringWriter sw = new StringWriter();
        getJspBody().invoke(sw);
        String content = sw.toString();

        JspWriter out = getJspContext().getOut();

        out.print("<span style=\"");
        if (color != null && !color.isEmpty()) {
            out.print("color: " + color + "; ");
        }
        if (bold) {
            out.print("font-weight: bold; ");
        }
        out.print("\">");
        out.print(content);
        out.print("</span>");
    }
}
```

2. Update the TLD:

```
<tag>
    <n>highlight</n>
    <tag-class>com.example.tags.HighlightTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
        <n>color</n>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <n>bold</n>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
        <type>boolean</type>
    </attribute>
</tag>
```

3. Use the Tag in a JSP:

```
<my:highlight color="blue" bold="true">
    This text will be blue and bold.
</my:highlight>

<my:highlight color="red">
    This text will be red but not bold.
</my:highlight>

<my:highlight bold="true">
    This text will be bold but not colored.
</my:highlight>

<p>
    <my:highlight color="${param.color}" bold="${param.bold eq 'true'}">
        This highlighting is controlled by URL parameters.
    </my:highlight>
</p>
```

Creating Tags with Nested Tags (Tag Collaboration)

Tags can collaborate with each other using the `findAncestorWithClass` method:

1. Create the Parent Tag:

```
package com.example.tags;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
import java.util.*;

public class TableTag extends SimpleTagSupport {

    private String border;
    private String cellpadding;
    private String cellspacing;
    private String styleClass;
    private List<String[]> rows = new ArrayList<>();

    public void setBorder(String border) {
        this.border = border;
    }

    public void setCellpadding(String cellpadding) {
        this.cellpadding = cellpadding;
    }
```

```
    public void setCellspacing(String cellspacing) {
        this.cellspacing = cellspacing;
    }

    public void setStyleClass(String styleClass) {
        this.styleClass = styleClass;
    }

    public void addRow(String[] cells) {
        rows.add(cells);
    }

    @Override
    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();

        // Start table
        out.print("<table");
        if (border != null) out.print(" border=\"" + border + "\"");
        if (cellpadding != null) out.print(" cellpadding=\"" + cellpadding +
"\"");
        if (cellspacing != null) out.print(" cellspacing=\"" + cellspacing +
"\"");
        if (styleClass != null) out.print(" class=\"" + styleClass + "\"");
        out.println(">");

        // Clear rows in case tag is reused
        rows.clear();

        // Process body (nested row tags will add rows)
        getJspBody().invoke(null);

        // Output all rows
        for (String[] row : rows) {
            out.println("  <tr>");
            for (String cell : row) {
                out.println("    <td>" + cell + "</td>");
            }
            out.println("  </tr>");
        }

        // End table
        out.println("</table>");
    }
}
```

2. Create the Child Tag:

```
package com.example.tags;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
```

```
import java.io.*;

public class RowTag extends SimpleTagSupport {

    @Override
    public void doTag() throws JspException, IOException {
        // Get body content
        StringWriter sw = new StringWriter();
        getJspBody().invoke(sw);
        String content = sw.toString();

        // Split content by commas
        String[] cells = content.split(",");

        // Find parent table tag
        TableTag parent = (TableTag) findAncestorWithClass(this, TableTag.class);

        if (parent == null) {
            throw new JspException("The row tag must be nested within a table
tag");
        }

        // Add row to parent
        parent.addRow(cells);
    }
}
```

3. Update the TLD:

```
<tag>
    <n>table</n>
    <tag-class>com.example.tags.TableTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
        <n>border</n>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <n>cellpadding</n>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <n>cellspacing</n>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <n>styleClass</n>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
```

```
    </attribute>
</tag>

<tag>
    <n>row</n>
    <tag-class>com.example.tags.RowTag</tag-class>
    <body-content>scriptless</body-content>
</tag>
```

4. Use the Tags in a JSP:

```
<my:table border="1" cellpadding="5" styleClass="data-table">
    <my:row>Name, Email, Age</my:row>
    <my:row>John Doe, john@example.com, 30</my:row>
    <my:row>Jane Smith, jane@example.com, 25</my:row>
    <my:row>Bob Johnson, bob@example.com, 45</my:row>
</my:table>
```

Tag Files

Tag files provide a simpler way to create custom tags using JSP syntax rather than Java code. They are similar to JSP includes but follow the custom tag calling convention.

1. Create a Tag File:

Create a file named `formatDate.tag` in the WEB-INF/tags directory:

```
<%@ tag body-content="empty" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ attribute name="date" required="true" type="java.util.Date" %>
<%@ attribute name="pattern" required="false" %>
<%@ attribute name="locale" required="false" %>

<% if (pattern == null) pattern = "yyyy-MM-dd HH:mm:ss"; %>

<c:if test="${not empty locale}">
    <fmt:setLocale value="${locale}" />
</c:if>

<fmt:formatDate value="${date}" pattern="${pattern}" />
```

2. Create a Tag File with Body:

Create a file named `card.tag` in the WEB-INF/tags directory:

```
<%@ tag body-content="scriptless" %>
<%@ attribute name="title" required="true" %>
```

```
<%@ attribute name="icon" required="false" %>
<%@ attribute name="backgroundColor" required="false" %>

<%
    if (backgroundColor == null || backgroundColor.isEmpty()) {
        backgroundColor = "#f8f9fa";
    }
%>

<div style="border: 1px solid #ddd; border-radius: 5px; overflow: hidden; margin-
bottom: 20px;">
    <div style="background-color: ${backgroundColor}; padding: 10px; border-
bottom: 1px solid #ddd;">
        <h3 style="margin: 0;">
            <% if (icon != null && !icon.isEmpty()) { %>
                <span class="${icon}"></span>
            <% } %>
            ${title}
        </h3>
    </div>
    <div style="padding: 15px;">
        <jsp:doBody />
    </div>
</div>
```

3. Use the Tag Files in a JSP:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Tag Files Demo</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.3/css/all.min.css">
</head>
<body>
    <h1>Tag Files Demonstration</h1>

    <h2>Format Date Example</h2>
    <p>Current date: <tags:formatDate date="<%= new java.util.Date() %>" /></p>
    <p>Custom format: <tags:formatDate date="<%= new java.util.Date() %>"
pattern="EEEE, MMMM d, yyyy" /></p>
    <p>French format: <tags:formatDate date="<%= new java.util.Date() %>"
pattern="d MMMM yyyy" locale="fr_FR" /></p>

    <h2>Card Component Example</h2>
    <tags:card title="User Profile" icon="fas fa-user" backgroundColor="#e3f2fd">
        <p><strong>Name:</strong> John Doe</p>
```

```
        <p><strong>Email:</strong> john@example.com</p>
        <p><strong>Role:</strong> Administrator</p>
        <button>Edit Profile</button>
    </tags:card>

    <tags:card title="System Status" icon="fas fa-server"
backgroundColor="#fff3cd">
        <p><strong>Status:</strong> Online</p>
        <p><strong>Uptime:</strong> 15 days, 7 hours</p>
        <p><strong>CPU Usage:</strong> 42%</p>
        <p><strong>Memory Usage:</strong> 3.2 GB / 8 GB</p>
    </tags:card>
</body>
</html>
```

Dynamic Attributes with Tag Files

You can create tag files that accept dynamic attributes:

1. Create a Tag File with Dynamic Attributes:

Create a file named `customElement.tag` in the WEB-INF/tags directory:

```
<%@ tag body-content="scriptless" dynamic-attributes="attrs" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ attribute name="element" required="true" %>

<${element}
<c:forEach var="entry" items="${attrs}">
    ${entry.key}="${entry.value}"
</c:forEach>
>
    <jsp:doBody />
</${element}>
```

2. Use the Tag in a JSP:

```
<tags:customElement element="div" id="content" class="container" style="padding:
20px;">
    <h2>Custom Element Example</h2>
    <p>This content is inside a custom div element.</p>
</tags:customElement>

<tags:customElement element="button" id="btn1" class="btn btn-primary"
onclick="alert('Clicked!');">
    Click Me
</tags:customElement>

<tags:customElement element="a" href="#" class="link" target="_blank">
```

```
    This is a link
</tags:customElement>
```

Custom Tag Best Practices

1. **Use Simple Tags**: Prefer SimpleTag over the classic Tag interface
2. **Use Tag Files for Simple Cases**: For presentation-oriented tags, use tag files
3. **Design for Reusability**:
    - Make attributes optional when possible
    - Provide sensible defaults
    - Document attributes and behavior
4. **Performance Considerations**:
    - Minimize expensive operations
    - Use buffer management carefully
    - Consider caching results when appropriate
5. **Error Handling**:
    - Validate attributes and inputs
    - Provide clear error messages
    - Handle exceptions gracefully
6. **Documentation**:
    - Add comments to your tag classes
    - Document attributes in the TLD
    - Provide usage examples
7. **Naming Conventions**:
    - Use consistent naming for tags and attributes
    - Follow Java Bean conventions for attribute setters
    - Use descriptive names that reflect tag functionality
8. **Testing**:
    - Test tags in isolation
    - Test tags with different attribute values
    - Test tags with invalid inputs

3.7 JSP Design Patterns

Design patterns help solve common architectural problems in JSP applications. They promote code organization, reusability, and maintainability.

MVC Pattern (Model-View-Controller)

The MVC pattern separates an application into three main components:

1. **Model**: Represents the data and business logic
2. **View**: Displays the data to the user (JSP pages)
3. **Controller**: Handles user input and updates the model (Servlets)

Example Implementation:

1. Model (User.java):

```
package com.example.model;

public class User {
    private int id;
    private String username;
    private String email;
    private String role;

    // Constructors
    public User() {}

    public User(int id, String username, String email, String role) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.role = role;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
}
```

2. Model (UserDAO.java):

```
package com.example.model;

import java.util.*;

public class UserDAO {
    private static Map<Integer, User> users = new HashMap<>();
```

```
    private static int nextId = 1;

    static {
        // Initialize with some sample data
        addUser(new User(nextId++, "john_doe", "john@example.com", "admin"));
        addUser(new User(nextId++, "jane_smith", "jane@example.com", "user"));
        addUser(new User(nextId++, "bob_johnson", "bob@example.com", "user"));
    }

    public static List<User> getAllUsers() {
        return new ArrayList<>(users.values());
    }

    public static User getUserById(int id) {
        return users.get(id);
    }

    public static User getUserByUsername(String username) {
        for (User user : users.values()) {
            if (user.getUsername().equals(username)) {
                return user;
            }
        }
        return null;
    }

    public static void addUser(User user) {
        if (user.getId() == 0) {
            user.setId(nextId++);
        }
        users.put(user.getId(), user);
    }

    public static void updateUser(User user) {
        users.put(user.getId(), user);
    }

    public static void deleteUser(int id) {
        users.remove(id);
    }
}
```

3. Controller (UserController.java):

```
package com.example.controller;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import com.example.model.*;
import java.util.*;
```

```
@WebServlet("/user/*")
public class UserController extends HttpServlet {

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

        String pathInfo = request.getPathInfo();

        if (pathInfo == null || pathInfo.equals("/")) {
            // List all users
            List<User> userList = UserDAO.getAllUsers();
            request.setAttribute("users", userList);
            request.getRequestDispatcher("/WEB-
INF/views/user/list.jsp").forward(request, response);

        } else if (pathInfo.equals("/add")) {
            // Show add form
            request.getRequestDispatcher("/WEB-
INF/views/user/form.jsp").forward(request, response);

        } else if (pathInfo.equals("/edit")) {
            // Show edit form
            int id = Integer.parseInt(request.getParameter("id"));
            User user = UserDAO.getUserById(id);
            request.setAttribute("user", user);
            request.getRequestDispatcher("/WEB-
INF/views/user/form.jsp").forward(request, response);

        } else if (pathInfo.startsWith("/view/")) {
            // View user details
            int id = Integer.parseInt(pathInfo.substring(6));
            User user = UserDAO.getUserById(id);
            request.setAttribute("user", user);
            request.getRequestDispatcher("/WEB-
INF/views/user/view.jsp").forward(request, response);

        } else if (pathInfo.equals("/delete")) {
            // Delete user
            int id = Integer.parseInt(request.getParameter("id"));
            UserDAO.deleteUser(id);
            response.sendRedirect(request.getContextPath() + "/user/");

        } else {
            // Handle 404
            response.sendError(HttpServletResponse.SC_NOT_FOUND);
        }
    }

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

```
        String pathInfo = request.getPathInfo();

        if (pathInfo.equals("/save")) {
            // Get form parameters
            String idParam = request.getParameter("id");
            String username = request.getParameter("username");
            String email = request.getParameter("email");
            String role = request.getParameter("role");

            User user;
            if (idParam == null || idParam.isEmpty()) {
                // Add new user
                user = new User();
            } else {
                // Update existing user
                int id = Integer.parseInt(idParam);
                user = UserDAO.getUserById(id);
            }

            user.setUsername(username);
            user.setEmail(email);
            user.setRole(role);

            UserDAO.addUser(user);

            response.sendRedirect(request.getContextPath() + "/user/");
        }
    }
}
```

4. View (list.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>User List</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
        .actions { width: 200px; }
    </style>
</head>
<body>
    <h1>User List</h1>

    <p><a href="${pageContext.request.contextPath}/user/add">Add New User</a></p>
```

```
    <table>
        <tr>
            <th>ID</th>
            <th>Username</th>
            <th>Email</th>
            <th>Role</th>
            <th class="actions">Actions</th>
        </tr>
        <c:forEach var="user" items="${users}">
            <tr>
                <td>${user.id}</td>
                <td>${user.username}</td>
                <td>${user.email}</td>
                <td>${user.role}</td>
                <td>
                    <a
href="${pageContext.request.contextPath}/user/view/${user.id}">View</a> |
                    <a href="${pageContext.request.contextPath}/user/edit?
id=${user.id}">Edit</a> |
                    <a href="${pageContext.request.contextPath}/user/delete?
id=${user.id}"
                       onclick="return confirm('Are you sure?')">Delete</a>
                </td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

5. View (form.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>${empty user ? 'Add User' : 'Edit User'}</title>
    <style>
        .form-group { margin-bottom: 15px; }
        label { display: inline-block; width: 100px; }
        input, select { padding: 5px; width: 300px; }
        button { padding: 8px 15px; background-color: #4CAF50; color: white;
border: none; cursor: pointer; }
    </style>
</head>
<body>
    <h1>${empty user ? 'Add User' : 'Edit User'}</h1>

    <form action="${pageContext.request.contextPath}/user/save" method="post">
```

```
        <c:if test="${not empty user}">
            <input type="hidden" name="id" value="${user.id}">
        </c:if>

        <div class="form-group">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username"
value="${user.username}" required>
        </div>

        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" value="${user.email}"
required>
        </div>

        <div class="form-group">
            <label for="role">Role:</label>
            <select id="role" name="role" required>
                <option value="">Select Role</option>
                <option value="admin" ${user.role == 'admin' ? 'selected' :
''}>Admin</option>
                <option value="user" ${user.role == 'user' ? 'selected' :
''}>User</option>
            </select>
        </div>

        <div class="form-group">
            <button type="submit">Save</button>
            <a href="${pageContext.request.contextPath}/user/">Cancel</a>
        </div>
    </form>
</body>
</html>
```

6. View (view.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>View User</title>
    <style>
        .user-info { border: 1px solid #ddd; padding: 20px; max-width: 500px;
margin: 0 auto; }
        .info-group { margin-bottom: 15px; }
        .label { font-weight: bold; }
    </style>
</head>
<body>
```

```
    <h1>User Details</h1>

    <div class="user-info">
        <div class="info-group">
            <span class="label">ID:</span> ${user.id}
        </div>
        <div class="info-group">
            <span class="label">Username:</span> ${user.username}
        </div>
        <div class="info-group">
            <span class="label">Email:</span> ${user.email}
        </div>
        <div class="info-group">
            <span class="label">Role:</span> ${user.role}
        </div>
    </div>

    <p>
        <a href="${pageContext.request.contextPath}/user/edit?
id=${user.id}">Edit</a> |
        <a href="${pageContext.request.contextPath}/user/">Back to List</a>
    </p>
</body>
</html>
```

Benefits of MVC:

- Clear separation of concerns
- Improved maintainability
- Easier testing
- Better code organization
- Reusable components

Front Controller Pattern

The Front Controller pattern provides a centralized entry point for handling requests:

1. Front Controller (FrontController.java):

```
package com.example.controller;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.util.*;

@WebServlet("*.do")
public class FrontController extends HttpServlet {

    private Map<String, Controller> controllers;
```

```
    @Override
    public void init() throws ServletException {
        controllers = new HashMap<>();
        controllers.put("user", new UserController());
        controllers.put("product", new ProductController());
        controllers.put("order", new OrderController());
        // Add more controllers as needed
    }

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

        // Extract action from URL
        String uri = request.getRequestURI();
        String contextPath = request.getContextPath();
        String path = uri.substring(contextPath.length());

        // Format: /controller/action.do
        // Example: /user/list.do

        String[] parts = path.split("/");
        if (parts.length < 3) {
            response.sendError(HttpServletResponse.SC_BAD_REQUEST, "Invalid URL
format");
            return;
        }

        String controllerName = parts[1];
        String actionWithExtension = parts[2];
        String action = actionWithExtension.substring(0,
actionWithExtension.length() - 3); // Remove .do

        // Get controller
        Controller controller = controllers.get(controllerName);
        if (controller == null) {
            response.sendError(HttpServletResponse.SC_NOT_FOUND, "Controller not
found: " + controllerName);
            return;
        }

        // Execute action
        String view = controller.execute(request, response, action);

        // Forward to view if not null
        if (view != null) {
            request.getRequestDispatcher(view).forward(request, response);
        }
    }

    // Controller interface
    public interface Controller {
```

```
        String execute(HttpServletRequest request, HttpServletResponse response,
String action)
                throws ServletException, IOException;
    }
}
```

2. Controller Implementation (UserController.java):

```
package com.example.controller;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.example.model.*;
import java.util.*;
import com.example.controller.FrontController.Controller;

public class UserController implements Controller {

    @Override
    public String execute(HttpServletRequest request, HttpServletResponse
response, String action)
            throws ServletException, IOException {

        switch (action) {
            case "list":
                return list(request, response);
            case "view":
                return view(request, response);
            case "add":
                return add(request, response);
            case "edit":
                return edit(request, response);
            case "save":
                return save(request, response);
            case "delete":
                return delete(request, response);
            default:
                response.sendError(HttpServletResponse.SC_NOT_FOUND, "Action not
found: " + action);
                return null;
        }
    }

    private String list(HttpServletRequest request, HttpServletResponse response)
{
        List<User> userList = UserDAO.getAllUsers();
        request.setAttribute("users", userList);
        return "/WEB-INF/views/user/list.jsp";
    }

    private String view(HttpServletRequest request, HttpServletResponse response)
```

```
{
        int id = Integer.parseInt(request.getParameter("id"));
        User user = UserDAO.getUserById(id);
        request.setAttribute("user", user);
        return "/WEB-INF/views/user/view.jsp";
    }

    private String add(HttpServletRequest request, HttpServletResponse response) {
        return "/WEB-INF/views/user/form.jsp";
    }

    private String edit(HttpServletRequest request, HttpServletResponse response)
{
        int id = Integer.parseInt(request.getParameter("id"));
        User user = UserDAO.getUserById(id);
        request.setAttribute("user", user);
        return "/WEB-INF/views/user/form.jsp";
    }

    private String save(HttpServletRequest request, HttpServletResponse response)
            throws IOException {

        String idParam = request.getParameter("id");
        String username = request.getParameter("username");
        String email = request.getParameter("email");
        String role = request.getParameter("role");

        User user;
        if (idParam == null || idParam.isEmpty()) {
            user = new User();
        } else {
            int id = Integer.parseInt(idParam);
            user = UserDAO.getUserById(id);
        }

        user.setUsername(username);
        user.setEmail(email);
        user.setRole(role);

        UserDAO.addUser(user);

        response.sendRedirect(request.getContextPath() + "/user/list.do");
        return null; // No forwarding as we've already redirected
    }

    private String delete(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {

        int id = Integer.parseInt(request.getParameter("id"));
        UserDAO.deleteUser(id);

        response.sendRedirect(request.getContextPath() + "/user/list.do");
        return null; // No forwarding as we've already redirected
```

```
    }
}
```

3. JSP View (with updated URLs):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>User List</title>
    <style>
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
        .actions { width: 200px; }
    </style>
</head>
<body>
    <h1>User List</h1>

    <p><a href="${pageContext.request.contextPath}/user/add.do">Add New User</a>
</p>

    <table>
        <tr>
            <th>ID</th>
            <th>Username</th>
            <th>Email</th>
            <th>Role</th>
            <th class="actions">Actions</th>
        </tr>
        <c:forEach var="user" items="${users}">
            <tr>
                <td>${user.id}</td>
                <td>${user.username}</td>
                <td>${user.email}</td>
                <td>${user.role}</td>
                <td>
                    <a href="${pageContext.request.contextPath}/user/view.do?
id=${user.id}">View</a> |
                    <a href="${pageContext.request.contextPath}/user/edit.do?
id=${user.id}">Edit</a> |
                    <a href="${pageContext.request.contextPath}/user/delete.do?
id=${user.id}"
                       onclick="return confirm('Are you sure?')">Delete</a>
                </td>
            </tr>
        </c:forEach>
    </table>
```

```
</body>
</html>
```

Benefits of Front Controller:

- Centralized request handling
- Consistent preprocessing/postprocessing
- Simplified navigation and request mapping
- Easier to implement cross-cutting concerns (security, logging)

Service Locator Pattern

The Service Locator pattern provides a centralized registry for services:

1. Service Locator (ServiceLocator.java):

```
package com.example.util;

import java.util.*;

public class ServiceLocator {

    private static final Map<String, Object> services = new HashMap<>();

    public static void register(String name, Object service) {
        services.put(name, service);
    }

    public static Object getService(String name) {
        return services.get(name);
    }

    @SuppressWarnings("unchecked")
    public static <T> T getService(String name, Class<T> type) {
        Object service = services.get(name);
        if (service != null && type.isInstance(service)) {
            return (T) service;
        }
        return null;
    }

    public static void unregister(String name) {
        services.remove(name);
    }
}
```

2. Initializing Services (ServiceInitializer.java):

```

```

```
package com.example.util;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import com.example.service.*;

@WebListener
public class ServiceInitializer implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        // Register services
        ServiceLocator.register("userService", new UserService());
        ServiceLocator.register("productService", new ProductService());
        ServiceLocator.register("orderService", new OrderService());
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        // Clean up resources if needed
    }
}
```

3. Using Services in a Servlet:

```
@WebServlet("/users")
public class UserServlet extends HttpServlet {

    private UserService userService;

    @Override
    public void init() throws ServletException {
        userService = ServiceLocator.getService("userService", UserService.class);
        if (userService == null) {
            throw new ServletException("UserService not available");
        }
    }

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

        List<User> users = userService.getAllUsers();
        request.setAttribute("users", users);
        request.getRequestDispatcher("/WEB-INF/views/users.jsp").forward(request,
response);
    }
}
```

Benefits of Service Locator:

- Decouples service consumers from service implementations
- Makes it easier to swap implementations
- Centralizes service configuration
- Simplifies testing with mock services

DAO Pattern (Data Access Object)

The DAO pattern separates data access logic from business logic:

1. DAO Interface (ProductDAO.java):

```
package com.example.dao;

import com.example.model.Product;
import java.util.List;

public interface ProductDAO {
    List<Product> findAll();
    Product findById(int id);
    List<Product> findByCategory(String category);
    void save(Product product);
    void update(Product product);
    void delete(int id);
}
```

2. DAO Implementation (JdbcProductDAO.java):

```
package com.example.dao.impl;

import com.example.dao.ProductDAO;
import com.example.model.Product;
import com.example.util.DBUtil;
import java.sql.*;
import java.util.*;

public class JdbcProductDAO implements ProductDAO {

    @Override
    public List<Product> findAll() {
        List<Product> products = new ArrayList<>();
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBUtil.getConnection();
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM products");
```

```
            while (rs.next()) {
                Product product = mapResultSetToProduct(rs);
                products.add(product);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DBUtil.closeResources(conn, stmt, rs);
        }

        return products;
    }

    @Override
    public Product findById(int id) {
        Product product = null;
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBUtil.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM products WHERE id = ?");
            stmt.setInt(1, id);
            rs = stmt.executeQuery();

            if (rs.next()) {
                product = mapResultSetToProduct(rs);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DBUtil.closeResources(conn, stmt, rs);
        }

        return product;
    }

    @Override
    public List<Product> findByCategory(String category) {
        List<Product> products = new ArrayList<>();
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBUtil.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM products WHERE category =
?");
            stmt.setString(1, category);
            rs = stmt.executeQuery();
```

```
            while (rs.next()) {
                Product product = mapResultSetToProduct(rs);
                products.add(product);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DBUtil.closeResources(conn, stmt, rs);
        }

        return products;
    }

    @Override
    public void save(Product product) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBUtil.getConnection();
            stmt = conn.prepareStatement(
                "INSERT INTO products (name, description, price, category) VALUES
(?, ?, ?, ?)",
                Statement.RETURN_GENERATED_KEYS
            );

            stmt.setString(1, product.getName());
            stmt.setString(2, product.getDescription());
            stmt.setDouble(3, product.getPrice());
            stmt.setString(4, product.getCategory());

            stmt.executeUpdate();

            // Get generated ID
            ResultSet generatedKeys = stmt.getGeneratedKeys();
            if (generatedKeys.next()) {
                product.setId(generatedKeys.getInt(1));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DBUtil.closeResources(conn, stmt, null);
        }
    }

    @Override
    public void update(Product product) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
```

```
            conn = DBUtil.getConnection();
            stmt = conn.prepareStatement(
                "UPDATE products SET name = ?, description = ?, price = ?,
category = ? WHERE id = ?"
            );

            stmt.setString(1, product.getName());
            stmt.setString(2, product.getDescription());
            stmt.setDouble(3, product.getPrice());
            stmt.setString(4, product.getCategory());
            stmt.setInt(5, product.getId());

            stmt.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DBUtil.closeResources(conn, stmt, null);
        }
    }

    @Override
    public void delete(int id) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBUtil.getConnection();
            stmt = conn.prepareStatement("DELETE FROM products WHERE id = ?");
            stmt.setInt(1, id);
            stmt.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DBUtil.closeResources(conn, stmt, null);
        }
    }

    private Product mapResultSetToProduct(ResultSet rs) throws SQLException {
        Product product = new Product();
        product.setId(rs.getInt("id"));
        product.setName(rs.getString("name"));
        product.setDescription(rs.getString("description"));
        product.setPrice(rs.getDouble("price"));
        product.setCategory(rs.getString("category"));
        return product;
    }
}
```

3. Database Utility Class (DBUtil.java):

```
package com.example.util;

import java.sql.*;

public class DBUtil {

    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/mydb";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "password";

    static {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
    }

    public static void closeResources(Connection conn, Statement stmt, ResultSet
rs) {
        try {
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

4. Using the DAO in a Service:

```
package com.example.service;

import com.example.dao.ProductDAO;
import com.example.dao.impl.JdbcProductDAO;
import com.example.model.Product;
import java.util.List;

public class ProductService {

    private ProductDAO productDAO;

    public ProductService() {
        productDAO = new JdbcProductDAO();
    }
```

```
    public List<Product> getAllProducts() {
        return productDAO.findAll();
    }

    public Product getProductById(int id) {
        return productDAO.findById(id);
    }

    public List<Product> getProductsByCategory(String category) {
        return productDAO.findByCategory(category);
    }

    public void saveProduct(Product product) {
        productDAO.save(product);
    }

    public void updateProduct(Product product) {
        productDAO.update(product);
    }

    public void deleteProduct(int id) {
        productDAO.delete(id);
    }
}
```

Benefits of DAO Pattern:

- Separates data access code from business logic
- Makes it easier to change data sources
- Promotes testability
- Enhances code organization

Business Delegate Pattern

The Business Delegate pattern decouples presentation tier from business tier:

1. Business Delegate (UserBusinessDelegate.java):

```
package com.example.delegate;

import com.example.model.User;
import com.example.service.UserService;
import java.util.List;

public class UserBusinessDelegate {

    private UserService userService;

    public UserBusinessDelegate() {
        userService = new UserService();
    }
```

```
    public List<User> getUsers() {
        return userService.getAllUsers();
    }

    public User getUser(int id) {
        return userService.getUserById(id);
    }

    public void saveUser(User user) {
        userService.saveUser(user);
    }

    public void updateUser(User user) {
        userService.updateUser(user);
    }

    public void deleteUser(int id) {
        userService.deleteUser(id);
    }
}
```

2. Using the Business Delegate in a Servlet:

```
@WebServlet("/users/*")
public class UserServlet extends HttpServlet {

    private UserBusinessDelegate delegate;

    @Override
    public void init() throws ServletException {
        delegate = new UserBusinessDelegate();
    }

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

        String pathInfo = request.getPathInfo();

        if (pathInfo == null || pathInfo.equals("/")) {
            // List all users
            List<User> users = delegate.getUsers();
            request.setAttribute("users", users);
            request.getRequestDispatcher("/WEB-
INF/views/user/list.jsp").forward(request, response);
        } else if (pathInfo.startsWith("/view/")) {
            // View user details
            int id = Integer.parseInt(pathInfo.substring(6));
            User user = delegate.getUser(id);
            request.setAttribute("user", user);
            request.getRequestDispatcher("/WEB-
```

```
INF/views/user/view.jsp").forward(request, response);
        }
        // Other actions...
    }
}
```

Benefits of Business Delegate:

- Reduces coupling between tiers
- Centralizes service access logic
- Enhances maintainability
- Simplifies client code

Composite View Pattern

The Composite View pattern combines multiple views into a single view:

1. Layout JSP (layout.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>${param.title}</title>
    <link rel="stylesheet"
href="${pageContext.request.contextPath}/css/style.css">
</head>
<body>
    <div class="container">
        <header>
            <jsp:include page="/WEB-INF/views/common/header.jsp" />
        </header>

        <nav>
            <jsp:include page="/WEB-INF/views/common/nav.jsp" />
        </nav>

        <div class="content">
            <jsp:include page="${param.content}" />
        </div>

        <footer>
            <jsp:include page="/WEB-INF/views/common/footer.jsp" />
        </footer>
    </div>
</body>
</html>
```

2. Header JSP (header.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<div class="header">
    <h1>My Web Application</h1>
    <p>Welcome ${not empty sessionScope.user ? sessionScope.user.username :
'Guest'}</p>
</div>
```

3. Navigation JSP (nav.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<ul class="nav">
    <li><a href="${pageContext.request.contextPath}/">Home</a></li>
    <li><a href="${pageContext.request.contextPath}/user/">Users</a></li>
    <li><a href="${pageContext.request.contextPath}/product/">Products</a></li>
    <li><a href="${pageContext.request.contextPath}/order/">Orders</a></li>

    <c:choose>
        <c:when test="${empty sessionScope.user}">
            <li class="right"><a
href="${pageContext.request.contextPath}/login">Login</a></li>
        </c:when>
        <c:otherwise>
            <li class="right"><a
href="${pageContext.request.contextPath}/logout">Logout</a></li>
        </c:otherwise>
    </c:choose>
</ul>
```

4. Footer JSP (footer.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<div class="footer">
    <p>&copy; 2023 My Web Application. All rights reserved.</p>
</div>
```

5. Content JSP (user-list.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

```
<h1>User List</h1>

<p><a href="${pageContext.request.contextPath}/user/add" class="btn">Add New
User</a></p>

<table>
    <tr>
        <th>ID</th>
        <th>Username</th>
        <th>Email</th>
        <th>Role</th>
        <th>Actions</th>
    </tr>
    <c:forEach var="user" items="${users}">
        <tr>
            <td>${user.id}</td>
            <td>${user.username}</td>
            <td>${user.email}</td>
            <td>${user.role}</td>
            <td>
                <a
href="${pageContext.request.contextPath}/user/view/${user.id}">View</a> |
                <a href="${pageContext.request.contextPath}/user/edit?
id=${user.id}">Edit</a> |
                <a href="${pageContext.request.contextPath}/user/delete?
id=${user.id}"
                   onclick="return confirm('Are you sure?')">Delete</a>
            </td>
        </tr>
    </c:forEach>
</table>
```

6. Using the Layout in a Servlet:

```
@WebServlet("/user/")
public class UserListServlet extends HttpServlet {

    private UserService userService;

    @Override
    public void init() throws ServletException {
        userService = new UserService();
    }

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

        List<User> users = userService.getAllUsers();
        request.setAttribute("users", users);

        request.getRequestDispatcher("/WEB-INF/views/layout.jsp?
```

```
        title=User+List&content=/WEB-INF/views/user/user-list.jsp")
                .forward(request, response);
        }
    }
```

Benefits of Composite View:

- Consistent look and feel
- Reuse of common elements
- Better code organization
- Simplified maintenance

Best Practices for JSP Design Patterns

1. **Choose the Right Pattern**:
    - MVC for overall application architecture
    - Front Controller for centralized request handling
    - DAO for data access
    - Composite View for consistent layout
2. **Combine Patterns**:
    - Patterns can be used together
    - For example, MVC + Front Controller + DAO + Composite View
3. **Keep Separation of Concerns**:
    - JSPs should focus on presentation
    - Servlets should handle request processing
    - Business logic should be in service classes
    - Data access should be in DAO classes
4. **Use Design Patterns Consistently**:
    - Apply patterns uniformly across the application
    - Document the patterns used
5. **Consider Performance**:
    - Some patterns add complexity and overhead
    - Balance maintainability with performance needs

3.8 Error Pages and Exception Handling

Proper error handling is essential for a good user experience. JSP provides several mechanisms for handling errors and exceptions.

Basic Error Page Configuration

The simplest way to define error pages is in the web.xml deployment descriptor:

```
<error-page>
    <error-code>404</error-code>
    <location>/WEB-INF/error/404.jsp</location>
</error-page>

<error-page>
    <error-code>500</error-code>
    <location>/WEB-INF/error/500.jsp</location>
</error-page>

<error-page>
    <exception-type>java.lang.NullPointerException</exception-type>
    <location>/WEB-INF/error/null-pointer.jsp</location>
</error-page>

<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/WEB-INF/error/exception.jsp</location>
</error-page>
```

You can define error pages for:

- Specific HTTP status codes (e.g., 404, 500)
- Specific exception types (e.g., NullPointerException)
- General exceptions (java.lang.Exception for all other exceptions)

Creating Custom Error Pages

Custom error pages should be informative but not reveal sensitive information:

1. 404 Error Page (404.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Page Not Found</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f5f5f5;
        }
        .error-container {
            max-width: 800px;
            margin: 40px auto;
            background-color: #fff;
```

```
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            text-align: center;
        }
        h1 {
            color: #d9534f;
            font-size: 36px;
            margin-bottom: 10px;
        }
        .error-code {
            font-size: 24px;
            color: #777;
            margin-bottom: 20px;
        }
        .error-message {
            margin: 30px 0;
            font-size: 18px;
        }
        .home-link {
            display: inline-block;
            padding: 10px 20px;
            background-color: #5bc0de;
            color: #fff;
            text-decoration: none;
            border-radius: 4px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <div class="error-container">
        <h1>Oops! Page Not Found</h1>
        <div class="error-code">Error 404</div>

        <div class="error-message">
            <p>The page you are looking for might have been removed, had its name
changed,
               or is temporarily unavailable.</p>
        </div>

        <p>Here are some helpful links:</p>
        <ul style="list-style-type: none; padding: 0;">
            <li><a href="${pageContext.request.contextPath}/">Home Page</a></li>
            <li><a href="#" onclick="history.back(); return false;">Go Back</a>
</li>
            <li><a href="${pageContext.request.contextPath}/contact">Contact
Support</a></li>
        </ul>

        <div style="margin-top: 30px;">
            <a href="${pageContext.request.contextPath}/" class="home-link">Return
to Homepage</a>
        </div>
```

```
    </div>
</body>
</html>
```

2. 500 Error Page (500.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Internal Server Error</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f5f5f5;
        }
        .error-container {
            max-width: 800px;
            margin: 40px auto;
            background-color: #fff;
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            text-align: center;
        }
        h1 {
            color: #d9534f;
            font-size: 36px;
            margin-bottom: 10px;
        }
        .error-code {
            font-size: 24px;
            color: #777;
            margin-bottom: 20px;
        }
        .error-message {
            margin: 30px 0;
            font-size: 18px;
        }
        .home-link {
            display: inline-block;
            padding: 10px 20px;
            background-color: #5bc0de;
            color: #fff;
            text-decoration: none;
            border-radius: 4px;
            font-weight: bold;
        }
```

```
        .error-details {
            margin-top: 30px;
            text-align: left;
            background-color: #f8f9fa;
            padding: 15px;
            border-radius: 4px;
        }
    </style>
</head>
<body>
    <div class="error-container">
        <h1>Internal Server Error</h1>
        <div class="error-code">Error 500</div>

        <div class="error-message">
            <p>Sorry, something went wrong on our servers.</p>
            <p>We're working to fix the issue and will be back shortly.</p>
        </div>

        <p>Here are some options:</p>
        <ul style="list-style-type: none; padding: 0;">
            <li><a href="${pageContext.request.contextPath}/">Home Page</a></li>
            <li><a href="#" onclick="history.back(); return false;">Go Back</a>
</li>
            <li><a href="${pageContext.request.contextPath}/contact">Contact
Support</a></li>
        </ul>

        <div style="margin-top: 30px;">
            <a href="${pageContext.request.contextPath}/" class="home-link">Return
to Homepage</a>
        </div>

        <%-- Show error details if in development mode --%>
        <% if (Boolean.TRUE.equals(application.getAttribute("developmentMode"))) {
%>
            <div class="error-details">
                <h3>Error Details (visible in development mode only):</h3>
                <p><strong>Error Time:</strong> <%= new java.util.Date() %></p>
                <p><strong>Requested URL:</strong>
${pageContext.errorData.requestURI}</p>
                <p><strong>Status Code:</strong>
${pageContext.errorData.statusCode}</p>
                <p><strong>Exception Type:</strong>
${pageContext.exception.getClass().name}</p>
                <p><strong>Exception Message:</strong>
${pageContext.exception.message}</p>

                <h4>Stack Trace:</h4>
                <pre>
<%
    if (exception != null) {
        java.io.StringWriter sw = new java.io.StringWriter();
        java.io.PrintWriter pw = new java.io.PrintWriter(sw);
```

```
            exception.printStackTrace(pw);
            out.println(sw.toString());
        }
    %>
                    </pre>
                </div>
            <% } %>
        </div>
    </body>
    </html>
```

3. Exception Error Page (exception.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Application Error</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f5f5f5;
        }
        .error-container {
            max-width: 800px;
            margin: 40px auto;
            background-color: #fff;
            padding: 30px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        }
        h1 {
            color: #d9534f;
        }
        .error-message {
            margin: 20px 0;
            padding: 15px;
            background-color: #f8f9fa;
            border-left: 5px solid #d9534f;
        }
        .error-details {
            margin-top: 30px;
            background-color: #f8f9fa;
            padding: 15px;
            border-radius: 4px;
            overflow-x: auto;
        }
        pre {
```

```
            margin: 0;
            white-space: pre-wrap;
        }
    </style>
</head>
<body>
    <div class="error-container">
        <h1>Application Error</h1>

        <div class="error-message">
            <p>An unexpected error occurred while processing your request.</p>
            <p>Our technical team has been notified. Please try again later.</p>
        </div>

        <p><a href="${pageContext.request.contextPath}/">Return to Home Page</a>
</p>

        <%-- Show error details if in development mode --%>
        <% if (Boolean.TRUE.equals(application.getAttribute("developmentMode"))) {
%>
            <div class="error-details">
                <h3>Error Details (visible in development mode only):</h3>
                <p><strong>Error Time:</strong> <%= new java.util.Date() %></p>
                <p><strong>Requested URL:</strong>
${pageContext.errorData.requestURI}</p>
                <p><strong>Exception Type:</strong>
${pageContext.exception.getClass().name}</p>
                <p><strong>Exception Message:</strong>
${pageContext.exception.message}</p>

                <h4>Stack Trace:</h4>
                <pre>
<%
    if (exception != null) {
        java.io.StringWriter sw = new java.io.StringWriter();
        java.io.PrintWriter pw = new java.io.PrintWriter(sw);
        exception.printStackTrace(pw);
        out.println(sw.toString());
    }
%>
                </pre>
            </div>
        <% } %>
    </div>
</body>
</html>
```

Using the isErrorPage Attribute

To access the implicit `exception` object, set `isErrorPage="true"` in the page directive:

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

This gives access to:

- The `exception` implicit object
- `pageContext.errorData` properties
  - `errorData.statusCode` - HTTP status code
  - `errorData.requestURI` - The requested URI
  - `errorData.servletName` - The servlet name
  - `errorData.throwable` - The exception object

The errorPage Attribute

You can specify an error page directly in a JSP using the `errorPage` attribute:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" errorPage="/WEB-INF/error/page-error.jsp" %>
```

If an exception occurs in this JSP, the container will automatically forward to the specified error page.

Programmatic Error Handling in Servlets

You can also handle errors programmatically in servlets:

```
@WebServlet("/products")
public class ProductServlet extends HttpServlet {

    private ProductService productService;

    @Override
    public void init() throws ServletException {
        productService = new ProductService();
    }

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

        try {
            List<Product> products = productService.getAllProducts();
            request.setAttribute("products", products);
            request.getRequestDispatcher("/WEB-INF/views/product/list.jsp").forward(request, response);

        } catch (Exception e) {
            // Log the exception
```

```
            getServletContext().log("Error in ProductServlet", e);

            // Store exception in request for the error page
            request.setAttribute("javax.servlet.error.exception", e);
            request.setAttribute("javax.servlet.error.message", e.getMessage());
            request.setAttribute("javax.servlet.error.request_uri",
request.getRequestURI());

            // Forward to error page
            request.getRequestDispatcher("/WEB-
INF/error/exception.jsp").forward(request, response);
        }
    }
}
```

Creating a Centralized Error Handler

For a more centralized approach, create an error handling servlet:

```
@WebServlet("/error-handler")
public class ErrorHandlerServlet extends HttpServlet {

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

        // Get error information
        Throwable throwable = (Throwable)
request.getAttribute("javax.servlet.error.exception");
        Integer statusCode = (Integer)
request.getAttribute("javax.servlet.error.status_code");
        String servletName = (String)
request.getAttribute("javax.servlet.error.servlet_name");
        String requestUri = (String)
request.getAttribute("javax.servlet.error.request_uri");
        String errorMessage = (String)
request.getAttribute("javax.servlet.error.message");

        // Log the error
        if (throwable != null) {
            getServletContext().log("Error in " + servletName, throwable);
        } else {
            getServletContext().log("Error status " + statusCode + " for " +
requestUri);
        }

        // Store information in request
        request.setAttribute("statusCode", statusCode);
        request.setAttribute("throwable", throwable);
        request.setAttribute("servletName", servletName);
```

```
            request.setAttribute("requestUri", requestUri);
            request.setAttribute("errorMessage", errorMessage);
            request.setAttribute("timestamp", new java.util.Date());

            // Choose appropriate error page
            String errorPage;
            if (statusCode != null) {
                switch (statusCode) {
                    case 404:
                        errorPage = "/WEB-INF/error/404.jsp";
                        break;
                    case 500:
                        errorPage = "/WEB-INF/error/500.jsp";
                        break;
                    default:
                        errorPage = "/WEB-INF/error/general.jsp";
                }
            } else if (throwable instanceof NullPointerException) {
                errorPage = "/WEB-INF/error/null-pointer.jsp";
            } else {
                errorPage = "/WEB-INF/error/exception.jsp";
            }

            // Forward to error page
            request.getRequestDispatcher(errorPage).forward(request, response);
        }
    }
```

Configure it in web.xml:

```
<error-page>
    <error-code>404</error-code>
    <location>/error-handler</location>
</error-page>

<error-page>
    <error-code>500</error-code>
    <location>/error-handler</location>
</error-page>

<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error-handler</location>
</error-page>
```

Creating Custom Exception Classes

For application-specific errors, create custom exception classes:

```
package com.example.exception;

public class ProductNotFoundException extends Exception {

    private int productId;

    public ProductNotFoundException(int productId) {
        super("Product not found with ID: " + productId);
        this.productId = productId;
    }

    public ProductNotFoundException(int productId, String message) {
        super(message);
        this.productId = productId;
    }

    public ProductNotFoundException(int productId, String message, Throwable
cause) {
        super(message, cause);
        this.productId = productId;
    }

    public int getProductId() {
        return productId;
    }
}
```

Then create a specific error page for it:

```
<error-page>
    <exception-type>com.example.exception.ProductNotFoundException</exception-
type>
    <location>/WEB-INF/error/product-not-found.jsp</location>
</error-page>
```

JSTL Error Handling

JSTL provides the `<c:catch>` tag for error handling:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<c:catch var="exception">
    <% int result = 10 / 0; %>
    <p>Result: <%= result %></p>
</c:catch>

<c:if test="${not empty exception}">
    <div class="error">
```

```
            <p>An error occurred: ${exception.message}</p>
        </div>
    </c:if>
```

Error Handling Best Practices

1. **Plan for Different Error Types**:
    - HTTP errors (404, 500, etc.)
    - Application exceptions
    - System exceptions
2. **Create User-Friendly Error Pages**:
    - Clear, non-technical explanations
    - Helpful next steps
    - Consistent styling with the rest of the application
3. **Log Errors Properly**:
    - Include stack traces for debugging
    - Include request information
    - Use proper log levels
4. **Handle Security Carefully**:
    - Don't expose sensitive information in production
    - Don't reveal system details
    - But provide enough information for debugging
5. **Use Custom Exceptions**:
    - Create meaningful exception types
    - Include contextual information
    - Handle them specifically
6. **Add Error Prevention**:
    - Validate input early
    - Use defensive programming
    - Check for null values
7. **Test Error Handling**:
    - Test all error scenarios
    - Verify error pages display correctly
    - Check that appropriate information is logged
8. **Different Modes for Development and Production**:
    - Show detailed errors in development

- Show friendly messages in production

4. JDBC Integration

4.1 Connection Pooling

Connection pooling is a technique used to improve performance by reusing database connections instead of creating new ones for each request.

Why Use Connection Pooling?

1. **Performance Improvement**:
   - Creating database connections is expensive (time and resources)
   - Reusing connections improves response times
   - Reduces database server load
2. **Resource Management**:
   - Limits the number of concurrent connections
   - Prevents resource exhaustion
   - Handles connection timeouts
3. **Connection Validation**:
   - Detects and replaces broken connections
   - Ensures connections remain valid

Common Connection Pool Libraries

Several libraries provide connection pooling for Java applications:

1. **Apache DBCP** (Database Connection Pool)
2. **C3P0**
3. **HikariCP** (fastest and most lightweight)
4. **Tomcat JDBC Connection Pool**

Implementing Connection Pooling with DBCP

1. Add DBCP Dependencies:

For Maven:

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-dbcp2</artifactId>
    <version>2.9.0</version>
</dependency>
```

2. Create a Connection Pool Manager:

```
package com.example.util;

import org.apache.commons.dbcp2.BasicDataSource;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

public class DBConnectionPool {

    private static BasicDataSource dataSource;

    static {
        try {
            // Load database driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Initialize the datasource
            dataSource = new BasicDataSource();
            dataSource.setUrl("jdbc:mysql://localhost:3306/mydb");
            dataSource.setUsername("root");
            dataSource.setPassword("password");

            // Connection pool settings
            dataSource.setInitialSize(5);  // Initial connections
            dataSource.setMaxTotal(20);    // Maximum connections
            dataSource.setMaxIdle(10);     // Maximum idle connections
            dataSource.setMinIdle(5);      // Minimum idle connections
            dataSource.setMaxWaitMillis(10000); // Wait time for connection

            // Connection validation
            dataSource.setValidationQuery("SELECT 1");
            dataSource.setTestOnBorrow(true);
            dataSource.setTestWhileIdle(true);
            dataSource.setTimeBetweenEvictionRunsMillis(60000); // Test idle
connections every minute

            // Log configurations
            System.out.println("Database connection pool initialized
successfully");

        } catch (ClassNotFoundException e) {
            System.err.println("Database driver not found: " + e.getMessage());
        }
    }

    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();
    }

    public static DataSource getDataSource() {
        return dataSource;
    }
```

```
    public static void close() throws SQLException {
        if (dataSource != null) {
            dataSource.close();
        }
    }

    // Method to get pool statistics (useful for monitoring)
    public static String getPoolStats() {
        return String.format(
            "Pool Stats: Active=%d, Idle=%d, Total=%d",
            dataSource.getNumActive(),
            dataSource.getNumIdle(),
            dataSource.getMaxTotal()
        );
    }
}
```

3. Using the Connection Pool:

```
import java.sql.*;
import java.util.*;
import com.example.util.DBConnectionPool;
import com.example.model.Product;

public class ProductDAO {

    public List<Product> getAllProducts() {
        List<Product> products = new ArrayList<>();
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Get connection from pool
            conn = DBConnectionPool.getConnection();

            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM products");

            while (rs.next()) {
                Product product = new Product();
                product.setId(rs.getInt("id"));
                product.setName(rs.getString("name"));
                product.setPrice(rs.getDouble("price"));
                product.setDescription(rs.getString("description"));
                products.add(product);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
```

```
            // Close resources but NOT the connection
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                // Return connection to the pool
                if (conn != null) conn.close(); // This doesn't actually close but
returns to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return products;
    }

    // Other DAO methods...
}
```

4. Configuring Connection Pool in web.xml with JNDI:

An alternative approach is to configure the connection pool in your application server and access it via JNDI:

```
<resource-ref>
    <description>Database Connection Pool</description>
    <res-ref-name>jdbc/MyDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

5. Tomcat Configuration (context.xml):

```
<Context>
    <Resource
        name="jdbc/MyDB"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="com.mysql.cj.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/mydb"
        username="root"
        password="password"
        maxTotal="20"
        maxIdle="10"
        maxWaitMillis="10000"
        initialSize="5"
        validationQuery="SELECT 1"
        testOnBorrow="true"
        testWhileIdle="true"
        timeBetweenEvictionRunsMillis="60000"
```

```
    />
</Context>
```

6. Accessing the JNDI DataSource:

```
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

public class DBUtil {

    private static DataSource dataSource;

    static {
        try {
            InitialContext context = new InitialContext();
            dataSource = (DataSource) context.lookup("java:comp/env/jdbc/MyDB");
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();
    }

    public static void closeResources(Connection conn, Statement stmt, ResultSet
rs) {
        try {
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            if (conn != null) conn.close(); // Returns to pool
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Implementing Connection Pooling with HikariCP

HikariCP is a high-performance JDBC connection pool:

1. Add HikariCP Dependencies:

```
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
```

```
    <version>5.0.1</version>
</dependency>
```

2. Create a Connection Pool Manager:

```
package com.example.util;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

public class HikariCPDataSource {

    private static HikariDataSource dataSource;

    static {
        try {
            // Configure HikariCP
            HikariConfig config = new HikariConfig();
            config.setJdbcUrl("jdbc:mysql://localhost:3306/mydb");
            config.setUsername("root");
            config.setPassword("password");
            config.setDriverClassName("com.mysql.cj.jdbc.Driver");

            // Connection pool settings
            config.setMaximumPoolSize(10);
            config.setMinimumIdle(5);
            config.setIdleTimeout(30000);
            config.setConnectionTimeout(30000);
            config.setMaxLifetime(1800000);

            // Connection test query
            config.setConnectionTestQuery("SELECT 1");

            // Pool name
            config.setPoolName("MyHikariCP");

            // Create the datasource
            dataSource = new HikariDataSource(config);

            System.out.println("HikariCP connection pool initialized
successfully");

        } catch (Exception e) {
            System.err.println("Error initializing HikariCP: " + e.getMessage());
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
```

```
        return dataSource.getConnection();
    }

    public static DataSource getDataSource() {
        return dataSource;
    }

    public static void close() {
        if (dataSource != null) {
            dataSource.close();
        }
    }
}
```

Connection Pooling Best Practices

1. **Pool Sizing**:
    - Start with small pools (5-10 connections)
    - Monitor and adjust based on load
    - Consider CPU cores and database limits
2. **Connection Validation**:
    - Always validate connections before use
    - Test idle connections periodically
    - Use lightweight validation queries
3. **Timeout Configuration**:
    - Set appropriate wait timeout
    - Configure connection lifetime
    - Set idle connection timeout
4. **Exception Handling**:
    - Handle connection acquisition failures
    - Implement retry mechanisms
    - Log connection issues
5. **Monitoring**:
    - Track pool statistics (active/idle connections)
    - Monitor connection wait times
    - Watch for pool exhaustion
6. **Closing Resources**:
    - Always close ResultSets and Statements
    - Return connections to the pool (conn.close())
    - Use try-with-resources for automatic closing

7. **Application Server Integration**:
    - Use container-managed pools when possible
    - Configure via JNDI for better management
    - Let the application server handle pool lifecycle
8. **Transaction Management**:
    - Be aware of transaction boundaries
    - Don't hold connections longer than necessary
    - Use connection-per-request pattern for web applications

4.2 Data Access Patterns

Data Access Patterns provide structured approaches to interacting with databases, improving code organization, maintainability, and reusability.

Data Access Object (DAO) Pattern

DAO separates data access logic from business logic:

1. Define a DAO Interface:

```
package com.example.dao;

import com.example.model.User;
import java.util.List;

public interface UserDAO {
    User findById(long id);
    List<User> findAll();
    List<User> findByDepartment(String department);
    void save(User user);
    void update(User user);
    void delete(long id);
}
```

2. Implement the DAO Interface:

```
package com.example.dao.impl;

import com.example.dao.UserDAO;
import com.example.model.User;
import com.example.util.DBConnectionPool;
import java.sql.*;
import java.util.*;

public class JdbcUserDAO implements UserDAO {

    @Override
```

```
    public User findById(long id) {
        User user = null;
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM users WHERE id = ?");
            stmt.setLong(1, id);
            rs = stmt.executeQuery();

            if (rs.next()) {
                user = mapResultSetToUser(rs);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return user;
    }

    @Override
    public List<User> findAll() {
        List<User> users = new ArrayList<>();
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM users");

            while (rs.next()) {
                users.add(mapResultSetToUser(rs));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return users;
    }

    @Override
    public List<User> findByDepartment(String department) {
        List<User> users = new ArrayList<>();
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM users WHERE department =
?");
            stmt.setString(1, department);
            rs = stmt.executeQuery();

            while (rs.next()) {
                users.add(mapResultSetToUser(rs));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return users;
    }

    @Override
    public void save(User user) {
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement(
                "INSERT INTO users (username, email, department, created_at)
VALUES (?, ?, ?, ?)",
                Statement.RETURN_GENERATED_KEYS
            );
```

```
            stmt.setString(1, user.getUsername());
            stmt.setString(2, user.getEmail());
            stmt.setString(3, user.getDepartment());
            stmt.setTimestamp(4, new Timestamp(System.currentTimeMillis()));

            int affectedRows = stmt.executeUpdate();

            if (affectedRows == 0) {
                throw new SQLException("Creating user failed, no rows affected.");
            }

            rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                user.setId(rs.getLong(1));
            } else {
                throw new SQLException("Creating user failed, no ID obtained.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void update(User user) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement(
                "UPDATE users SET username = ?, email = ?, department = ? WHERE id
= ?"
            );

            stmt.setString(1, user.getUsername());
            stmt.setString(2, user.getEmail());
            stmt.setString(3, user.getDepartment());
            stmt.setLong(4, user.getId());

            stmt.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
```

```
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void delete(long id) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("DELETE FROM users WHERE id = ?");
            stmt.setLong(1, id);
            stmt.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    private User mapResultSetToUser(ResultSet rs) throws SQLException {
        User user = new User();
        user.setId(rs.getLong("id"));
        user.setUsername(rs.getString("username"));
        user.setEmail(rs.getString("email"));
        user.setDepartment(rs.getString("department"));
        user.setCreatedAt(rs.getTimestamp("created_at"));
        return user;
    }
}
```

3. Create a DAO Factory:

```
package com.example.dao;

import com.example.dao.impl.*;

public class DAOFactory {

    // Singleton instance
```

```
    private static DAOFactory instance;

    // Private constructor
    private DAOFactory() {}

    // Get instance
    public static synchronized DAOFactory getInstance() {
        if (instance == null) {
            instance = new DAOFactory();
        }
        return instance;
    }

    // Get DAO implementations
    public UserDAO getUserDAO() {
        return new JdbcUserDAO();
    }

    public ProductDAO getProductDAO() {
        return new JdbcProductDAO();
    }

    public OrderDAO getOrderDAO() {
        return new JdbcOrderDAO();
    }

    // Add more DAOs as needed
}
```

4. Using the DAO in a Service:

```
package com.example.service;

import com.example.dao.DAOFactory;
import com.example.dao.UserDAO;
import com.example.model.User;
import java.util.List;

public class UserService {

    private UserDAO userDAO;

    public UserService() {
        // Get DAO from factory
        userDAO = DAOFactory.getInstance().getUserDAO();
    }

    public User getUserById(long id) {
        return userDAO.findById(id);
    }

    public List<User> getAllUsers() {
```

```
        return userDAO.findAll();
    }

    public List<User> getUsersByDepartment(String department) {
        return userDAO.findByDepartment(department);
    }

    public void createUser(User user) {
        // Validate user data
        if (user.getUsername() == null || user.getUsername().trim().isEmpty()) {
            throw new IllegalArgumentException("Username cannot be empty");
        }
        if (user.getEmail() == null || user.getEmail().trim().isEmpty()) {
            throw new IllegalArgumentException("Email cannot be empty");
        }

        // Save user
        userDAO.save(user);
    }

    public void updateUser(User user) {
        // Validate user data
        if (user.getId() <= 0) {
            throw new IllegalArgumentException("Invalid user ID");
        }
        if (user.getUsername() == null || user.getUsername().trim().isEmpty()) {
            throw new IllegalArgumentException("Username cannot be empty");
        }
        if (user.getEmail() == null || user.getEmail().trim().isEmpty()) {
            throw new IllegalArgumentException("Email cannot be empty");
        }

        // Update user
        userDAO.update(user);
    }

    public void deleteUser(long id) {
        userDAO.delete(id);
    }
}
```

Repository Pattern

The Repository Pattern is similar to DAO but more domain-focused:

1. Define a Repository Interface:

```
package com.example.repository;

import com.example.model.Product;
import java.util.List;
```

```
public interface ProductRepository {
    Product findById(long id);
    List<Product> findAll();
    List<Product> findByCategory(String category);
    List<Product> findByPriceRange(double minPrice, double maxPrice);
    void save(Product product);
    void update(Product product);
    void delete(long id);
}
```

2. Implement the Repository:

```
package com.example.repository.impl;

import com.example.model.Product;
import com.example.repository.ProductRepository;
import com.example.util.DBConnectionPool;
import java.sql.*;
import java.util.*;

public class JdbcProductRepository implements ProductRepository {

    @Override
    public Product findById(long id) {
        // Implementation similar to DAO
    }

    @Override
    public List<Product> findAll() {
        // Implementation similar to DAO
    }

    @Override
    public List<Product> findByCategory(String category) {
        // Implementation similar to DAO
    }

    @Override
    public List<Product> findByPriceRange(double minPrice, double maxPrice) {
        List<Product> products = new ArrayList<>();
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM products WHERE price
BETWEEN ? AND ?");
            stmt.setDouble(1, minPrice);
            stmt.setDouble(2, maxPrice);
            rs = stmt.executeQuery();
```

```
            while (rs.next()) {
                products.add(mapResultSetToProduct(rs));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return products;
    }

    @Override
    public void save(Product product) {
        // Implementation similar to DAO
    }

    @Override
    public void update(Product product) {
        // Implementation similar to DAO
    }

    @Override
    public void delete(long id) {
        // Implementation similar to DAO
    }

    private Product mapResultSetToProduct(ResultSet rs) throws SQLException {
        Product product = new Product();
        product.setId(rs.getLong("id"));
        product.setName(rs.getString("name"));
        product.setDescription(rs.getString("description"));
        product.setPrice(rs.getDouble("price"));
        product.setCategory(rs.getString("category"));
        product.setInStock(rs.getBoolean("in_stock"));
        return product;
    }
}
```

Active Record Pattern

The Active Record pattern combines domain model and data access logic:

```java
package com.example.model;

import com.example.util.DBConnectionPool;
import java.sql.*;
import java.util.*;

public class Order {

    private long id;
    private long userId;
    private Date orderDate;
    private String status;
    private double totalAmount;

    // Constructors, getters, and setters

    // Data access methods

    public static Order findById(long id) {
        Order order = null;
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM orders WHERE id = ?");
            stmt.setLong(1, id);
            rs = stmt.executeQuery();

            if (rs.next()) {
                order = new Order();
                order.setId(rs.getLong("id"));
                order.setUserId(rs.getLong("user_id"));
                order.setOrderDate(rs.getDate("order_date"));
                order.setStatus(rs.getString("status"));
                order.setTotalAmount(rs.getDouble("total_amount"));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return order;
    }
```

```
    public static List<Order> findByUserId(long userId) {
        List<Order> orders = new ArrayList<>();
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM orders WHERE user_id =
?");
            stmt.setLong(1, userId);
            rs = stmt.executeQuery();

            while (rs.next()) {
                Order order = new Order();
                order.setId(rs.getLong("id"));
                order.setUserId(rs.getLong("user_id"));
                order.setOrderDate(rs.getDate("order_date"));
                order.setStatus(rs.getString("status"));
                order.setTotalAmount(rs.getDouble("total_amount"));
                orders.add(order);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return orders;
    }

    public void save() {
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();

            if (id == 0) {
                // Insert new order
                stmt = conn.prepareStatement(
                    "INSERT INTO orders (user_id, order_date, status,
total_amount) VALUES (?, ?, ?, ?)",
                    Statement.RETURN_GENERATED_KEYS
                );
```

```
                stmt.setLong(1, userId);
                stmt.setDate(2, new java.sql.Date(orderDate.getTime()));
                stmt.setString(3, status);
                stmt.setDouble(4, totalAmount);

                stmt.executeUpdate();

                rs = stmt.getGeneratedKeys();
                if (rs.next()) {
                    this.id = rs.getLong(1);
                }

            } else {
                // Update existing order
                stmt = conn.prepareStatement(
                    "UPDATE orders SET user_id = ?, order_date = ?, status = ?,
total_amount = ? WHERE id = ?"
                );

                stmt.setLong(1, userId);
                stmt.setDate(2, new java.sql.Date(orderDate.getTime()));
                stmt.setString(3, status);
                stmt.setDouble(4, totalAmount);
                stmt.setLong(5, id);

                stmt.executeUpdate();
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void delete() {
        if (id == 0) return;

        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("DELETE FROM orders WHERE id = ?");
            stmt.setLong(1, id);
            stmt.executeUpdate();
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                try {
                    if (stmt != null) stmt.close();
                    if (conn != null) conn.close(); // Return to pool
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }

        // Additional methods for order-specific logic
    }
```

Data Mapper Pattern

The Data Mapper pattern separates domain objects from database operations:

1. Define a Mapper Interface:

```
package com.example.mapper;

import java.util.List;

public interface DataMapper<T, K> {
    T findById(K id);
    List<T> findAll();
    void insert(T entity);
    void update(T entity);
    void delete(K id);
}
```

2. Implement a Concrete Mapper:

```
package com.example.mapper.impl;

import com.example.mapper.DataMapper;
import com.example.model.Customer;
import com.example.util.DBConnectionPool;
import java.sql.*;
import java.util.*;

public class CustomerMapper implements DataMapper<Customer, Long> {

    @Override
    public Customer findById(Long id) {
        Customer customer = null;
        Connection conn = null;
```

```
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM customers WHERE id = ?");
            stmt.setLong(1, id);
            rs = stmt.executeQuery();

            if (rs.next()) {
                customer = mapRow(rs);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return customer;
    }

    @Override
    public List<Customer> findAll() {
        List<Customer> customers = new ArrayList<>();
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM customers");

            while (rs.next()) {
                customers.add(mapRow(rs));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
```

```
        }

        return customers;
    }

    @Override
    public void insert(Customer customer) {
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement(
                "INSERT INTO customers (name, email, phone, address) VALUES (?, ?,
?, ?)",
                Statement.RETURN_GENERATED_KEYS
            );

            stmt.setString(1, customer.getName());
            stmt.setString(2, customer.getEmail());
            stmt.setString(3, customer.getPhone());
            stmt.setString(4, customer.getAddress());

            stmt.executeUpdate();

            rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                customer.setId(rs.getLong(1));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void update(Customer customer) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement(
                "UPDATE customers SET name = ?, email = ?, phone = ?, address = ?
WHERE id = ?"
```

```
            );

            stmt.setString(1, customer.getName());
            stmt.setString(2, customer.getEmail());
            stmt.setString(3, customer.getPhone());
            stmt.setString(4, customer.getAddress());
            stmt.setLong(5, customer.getId());

            stmt.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void delete(Long id) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("DELETE FROM customers WHERE id = ?");
            stmt.setLong(1, id);
            stmt.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    // Additional finder methods
    public List<Customer> findByName(String name) {
        List<Customer> customers = new ArrayList<>();
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
```

```
            stmt = conn.prepareStatement("SELECT * FROM customers WHERE name LIKE
?");
            stmt.setString(1, "%" + name + "%");
            rs = stmt.executeQuery();

            while (rs.next()) {
                customers.add(mapRow(rs));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return customers;
    }

    private Customer mapRow(ResultSet rs) throws SQLException {
        Customer customer = new Customer();
        customer.setId(rs.getLong("id"));
        customer.setName(rs.getString("name"));
        customer.setEmail(rs.getString("email"));
        customer.setPhone(rs.getString("phone"));
        customer.setAddress(rs.getString("address"));
        return customer;
    }
}
```

Table Data Gateway Pattern

The Table Data Gateway pattern provides a gateway to the database table:

```
package com.example.gateway;

import com.example.util.DBConnectionPool;
import java.sql.*;
import java.util.*;

public class ProductGateway {

    public Map<String, Object> findById(long id) {
        Map<String, Object> result = null;
        Connection conn = null;
        PreparedStatement stmt = null;
```

```
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM products WHERE id = ?");
            stmt.setLong(1, id);
            rs = stmt.executeQuery();

            if (rs.next()) {
                result = new HashMap<>();
                result.put("id", rs.getLong("id"));
                result.put("name", rs.getString("name"));
                result.put("description", rs.getString("description"));
                result.put("price", rs.getDouble("price"));
                result.put("category", rs.getString("category"));
                result.put("in_stock", rs.getBoolean("in_stock"));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return result;
    }

    public List<Map<String, Object>> findAll() {
        List<Map<String, Object>> results = new ArrayList<>();
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM products");

            while (rs.next()) {
                Map<String, Object> row = new HashMap<>();
                row.put("id", rs.getLong("id"));
                row.put("name", rs.getString("name"));
                row.put("description", rs.getString("description"));
                row.put("price", rs.getDouble("price"));
                row.put("category", rs.getString("category"));
                row.put("in_stock", rs.getBoolean("in_stock"));
                results.add(row);
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return results;
    }

    public List<Map<String, Object>> findByCategory(String category) {
        List<Map<String, Object>> results = new ArrayList<>();
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("SELECT * FROM products WHERE category =
?");
            stmt.setString(1, category);
            rs = stmt.executeQuery();

            while (rs.next()) {
                Map<String, Object> row = new HashMap<>();
                row.put("id", rs.getLong("id"));
                row.put("name", rs.getString("name"));
                row.put("description", rs.getString("description"));
                row.put("price", rs.getDouble("price"));
                row.put("category", rs.getString("category"));
                row.put("in_stock", rs.getBoolean("in_stock"));
                results.add(row);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return results;
    }
```

```
    public long insert(Map<String, Object> data) {
        long newId = 0;
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement(
                "INSERT INTO products (name, description, price, category,
in_stock) VALUES (?, ?, ?, ?, ?)",
                Statement.RETURN_GENERATED_KEYS
            );

            stmt.setString(1, (String) data.get("name"));
            stmt.setString(2, (String) data.get("description"));
            stmt.setDouble(3, (Double) data.get("price"));
            stmt.setString(4, (String) data.get("category"));
            stmt.setBoolean(5, (Boolean) data.get("in_stock"));

            stmt.executeUpdate();

            rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                newId = rs.getLong(1);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return newId;
    }

    public boolean update(Map<String, Object> data) {
        Connection conn = null;
        PreparedStatement stmt = null;
        boolean success = false;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement(
                "UPDATE products SET name = ?, description = ?, price = ?,
category = ?, in_stock = ? WHERE id = ?"
            );
```

```
            stmt.setString(1, (String) data.get("name"));
            stmt.setString(2, (String) data.get("description"));
            stmt.setDouble(3, (Double) data.get("price"));
            stmt.setString(4, (String) data.get("category"));
            stmt.setBoolean(5, (Boolean) data.get("in_stock"));
            stmt.setLong(6, (Long) data.get("id"));

            int rowsAffected = stmt.executeUpdate();
            success = (rowsAffected > 0);

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return success;
    }

    public boolean delete(long id) {
        Connection conn = null;
        PreparedStatement stmt = null;
        boolean success = false;

        try {
            conn = DBConnectionPool.getConnection();
            stmt = conn.prepareStatement("DELETE FROM products WHERE id = ?");
            stmt.setLong(1, id);

            int rowsAffected = stmt.executeUpdate();
            success = (rowsAffected > 0);

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return success;
    }
}
```

Data Access Patterns Comparison

Pattern	Description	Pros	Cons	Best For
DAO	Separates data access logic from business logic	Clear separation of concerns, easy to test	Can lead to many small classes	Most applications
Repository	Domain-focused data access	More aligned with domain model	Similar to DAO but more specific	Domain-driven design
Active Record	Domain model contains data access logic	Simple for CRUD operations	Tight coupling between model and DB	Simple applications
Data Mapper	Maps domain objects to database	Complete separation of domain and DB	More complex	Complex domain models
Table Data Gateway	Single class per table for data access	Simple table operations	Not object-oriented	Simple database operations

Best Practices for Data Access

1. **Use Prepared Statements**:
    - Prevents SQL injection
    - Improves performance through statement caching
    - Makes code cleaner and more maintainable
2. **Handle Resources Properly**:
    - Always close Connections, Statements, and ResultSets
    - Use try-with-resources for automatic cleanup
    - Return connections to the pool
3. **Error Handling**:
    - Catch and handle SQLExceptions appropriately
    - Log exception details for debugging
    - Provide meaningful error messages to users
4. **Use Transactions When Necessary**:
    - Maintain data integrity
    - Group related operations
    - Handle transaction boundaries properly
5. **Optimize Database Access**:
    - Limit the amount of data retrieved
    - Use appropriate indexes

- Optimize SQL queries

6. **Separation of Concerns**:
   - Keep data access separate from business logic
   - Use appropriate design patterns
   - Make code easier to maintain and test

7. **Use Connection Pooling**:
   - Improve performance
   - Manage resources effectively
   - Handle peak loads

4.3 Transaction Management

Transaction management ensures data integrity by grouping related operations that should succeed or fail as a unit.

Understanding Transactions

A transaction is a sequence of operations that should be executed as a single unit. It has four key properties (ACID):

1. **Atomicity**: All operations in a transaction succeed or fail together
2. **Consistency**: A transaction brings the database from one valid state to another
3. **Isolation**: Transactions execute as if they were the only ones accessing the database
4. **Durability**: Once a transaction is committed, its effects persist

JDBC Transaction Management

JDBC provides basic transaction management through the Connection interface:

1. Simple Transaction Example:

```
public void transferFunds(long fromAccountId, long toAccountId, double amount) {
    Connection conn = null;
    PreparedStatement withdrawStmt = null;
    PreparedStatement depositStmt = null;

    try {
        // Get connection from pool
        conn = DBConnectionPool.getConnection();

        // Disable auto-commit
        conn.setAutoCommit(false);

        // Prepare statements
        withdrawStmt = conn.prepareStatement(
            "UPDATE accounts SET balance = balance - ? WHERE id = ? AND balance >=
?"
```

```
            );
            depositStmt = conn.prepareStatement(
                "UPDATE accounts SET balance = balance + ? WHERE id = ?"
            );

            // Set parameters for withdrawal
            withdrawStmt.setDouble(1, amount);
            withdrawStmt.setLong(2, fromAccountId);
            withdrawStmt.setDouble(3, amount); // Ensure sufficient funds

            // Execute withdrawal
            int rowsAffected = withdrawStmt.executeUpdate();

            // Check if withdrawal was successful
            if (rowsAffected == 0) {
                // Rollback transaction
                conn.rollback();
                throw new InsufficientFundsException("Insufficient funds in account: "
+ fromAccountId);
            }

            // Set parameters for deposit
            depositStmt.setDouble(1, amount);
            depositStmt.setLong(2, toAccountId);

            // Execute deposit
            depositStmt.executeUpdate();

            // Commit transaction
            conn.commit();

        } catch (SQLException e) {
            // Rollback transaction on error
            try {
                if (conn != null) {
                    conn.rollback();
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
            }

            throw new RuntimeException("Transaction failed: " + e.getMessage(), e);

        } finally {
            // Reset auto-commit and close resources
            try {
                if (conn != null) {
                    conn.setAutoCommit(true);
                }

                if (withdrawStmt != null) withdrawStmt.close();
                if (depositStmt != null) depositStmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

Transaction Isolation Levels

JDBC supports different transaction isolation levels:

```
// Set isolation level
conn.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
```

Available isolation levels:

1. **TRANSACTION_READ_UNCOMMITTED**:
    - Allows "dirty reads" (uncommitted changes from other transactions)
    - Highest performance, lowest isolation
2. **TRANSACTION_READ_COMMITTED**:
    - Prevents dirty reads
    - Default in many databases
3. **TRANSACTION_REPEATABLE_READ**:
    - Prevents dirty reads and non-repeatable reads
    - Same rows read twice in a transaction return the same values
4. **TRANSACTION_SERIALIZABLE**:
    - Highest isolation level
    - Prevents dirty reads, non-repeatable reads, and phantom reads
    - Lowest performance

Transaction Manager Pattern

For more complex applications, implement a transaction manager:

```
package com.example.util;

import java.sql.Connection;
import java.sql.SQLException;

public class TransactionManager {

    private static final ThreadLocal<Connection> connectionHolder = new
ThreadLocal<>();
```

```
    // Begin a transaction
    public static void beginTransaction() throws SQLException {
        Connection conn = connectionHolder.get();

        if (conn == null) {
            conn = DBConnectionPool.getConnection();
            conn.setAutoCommit(false);
            connectionHolder.set(conn);
        }
    }

    // Get the current transaction connection
    public static Connection getConnection() throws SQLException {
        Connection conn = connectionHolder.get();

        if (conn == null) {
            throw new SQLException("No active transaction");
        }

        return conn;
    }

    // Commit the transaction
    public static void commitTransaction() throws SQLException {
        Connection conn = connectionHolder.get();

        if (conn != null) {
            try {
                conn.commit();
            } finally {
                cleanup();
            }
        }
    }

    // Rollback the transaction
    public static void rollbackTransaction() {
        Connection conn = connectionHolder.get();

        if (conn != null) {
            try {
                conn.rollback();
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                cleanup();
            }
        }
    }

    // Clean up resources
    private static void cleanup() {
        Connection conn = connectionHolder.get();
```

```
            if (conn != null) {
                try {
                    conn.setAutoCommit(true);
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                } finally {
                    connectionHolder.remove();
                }
            }
        }
    }
```

Using the Transaction Manager:

```
public void transferFunds(long fromAccountId, long toAccountId, double amount) {
    try {
        // Begin transaction
        TransactionManager.beginTransaction();

        // Get connection
        Connection conn = TransactionManager.getConnection();

        // Perform withdrawal
        PreparedStatement withdrawStmt = conn.prepareStatement(
            "UPDATE accounts SET balance = balance - ? WHERE id = ? AND balance >=
?"
        );
        withdrawStmt.setDouble(1, amount);
        withdrawStmt.setLong(2, fromAccountId);
        withdrawStmt.setDouble(3, amount);

        int rowsAffected = withdrawStmt.executeUpdate();
        withdrawStmt.close();

        if (rowsAffected == 0) {
            TransactionManager.rollbackTransaction();
            throw new InsufficientFundsException("Insufficient funds in account: "
+ fromAccountId);
        }

        // Perform deposit
        PreparedStatement depositStmt = conn.prepareStatement(
            "UPDATE accounts SET balance = balance + ? WHERE id = ?"
        );
        depositStmt.setDouble(1, amount);
        depositStmt.setLong(2, toAccountId);
        depositStmt.executeUpdate();
        depositStmt.close();

        // Commit transaction
        TransactionManager.commitTransaction();
```

```
    } catch (SQLException e) {
        // Rollback on error
        TransactionManager.rollbackTransaction();
        throw new RuntimeException("Transaction failed: " + e.getMessage(), e);
    }
}
```

Declarative Transaction Management

For declarative transaction management, you can use a filter or interceptor:

Transaction Filter:

```
package com.example.filter;

import com.example.util.TransactionManager;
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;
import java.sql.SQLException;

@WebFilter("/transaction/*")
public class TransactionFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        try {
            // Begin transaction
            TransactionManager.beginTransaction();

            // Process the request
            chain.doFilter(request, response);

            // Commit transaction if no exception
            TransactionManager.commitTransaction();

        } catch (SQLException e) {
            // Handle SQL exception
            TransactionManager.rollbackTransaction();
            throw new ServletException("Transaction failed", e);
        } catch (Exception e) {
            // Handle other exceptions
            TransactionManager.rollbackTransaction();
            throw e;
        }
    }
```

```
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

Transaction Best Practices

1. **Keep Transactions Short**:
    - Minimize the duration of transactions
    - Don't include user interactions in transactions
    - Avoid long-running transactions
2. **Handle Exceptions Properly**:
    - Always rollback on exceptions
    - Catch and handle specific exceptions
    - Log transaction errors
3. **Choose Appropriate Isolation Levels**:
    - Understand the tradeoffs between isolation and performance
    - Use the minimum isolation level needed
    - Be aware of database-specific behaviors
4. **Be Aware of Connection Pooling Interactions**:
    - Don't close connections within transactions
    - Return connections to the pool after transaction completion
    - Use try-with-resources for automatic cleanup
5. **Watch for Deadlocks**:
    - Access resources in a consistent order
    - Keep transactions small
    - Set appropriate timeouts
6. **Consider Using JTA for Distributed Transactions**:
    - For transactions spanning multiple resources
    - When working with multiple databases
    - For transactions involving JMS or other resources
7. **Test Transaction Scenarios**:
    - Test both success and failure scenarios

- Verify that rollbacks work correctly
- Test with concurrent users

4.4 PreparedStatements and CallableStatements

PreparedStatements and CallableStatements provide secure and efficient ways to execute SQL queries and stored procedures.

PreparedStatement

PreparedStatement is a precompiled SQL statement that can be executed multiple times with different parameters.

Benefits of PreparedStatement:

1. **SQL Injection Prevention**:
   - Parameters are properly escaped
   - Prevents attackers from manipulating SQL
2. **Performance Improvements**:
   - Precompiled statements are faster
   - Database can cache execution plans
   - Efficient for repeated execution
3. **Convenience**:
   - Easier parameter management
   - Automatic type conversion
   - Simpler code

Using PreparedStatement:

```
public List<Product> findProductsByCategory(String category, double minPrice) {
    List<Product> products = new ArrayList<>();
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;

    try {
        conn = DBConnectionPool.getConnection();

        // Create prepared statement with parameters
        stmt = conn.prepareStatement(
            "SELECT * FROM products WHERE category = ? AND price >= ? ORDER BY
name"
        );

        // Set parameters
        stmt.setString(1, category);    // First parameter
        stmt.setDouble(2, minPrice);    // Second parameter
```

```
            // Execute query
            rs = stmt.executeQuery();

            // Process results
            while (rs.next()) {
                Product product = new Product();
                product.setId(rs.getLong("id"));
                product.setName(rs.getString("name"));
                product.setDescription(rs.getString("description"));
                product.setPrice(rs.getDouble("price"));
                product.setCategory(rs.getString("category"));
                products.add(product);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return products;
    }
```

Parameter Setting Methods:

```
// String parameters
stmt.setString(1, "Electronics");

// Numeric parameters
stmt.setInt(2, 100);
stmt.setLong(3, 1234567890L);
stmt.setDouble(4, 99.99);
stmt.setBigDecimal(5, new BigDecimal("999.99"));

// Date and time parameters
stmt.setDate(6, java.sql.Date.valueOf("2023-01-15"));
stmt.setTime(7, java.sql.Time.valueOf("14:30:00"));
stmt.setTimestamp(8, java.sql.Timestamp.valueOf("2023-01-15 14:30:00"));

// Boolean parameters
stmt.setBoolean(9, true);

// Null parameters
stmt.setNull(10, java.sql.Types.VARCHAR);
```

```
// Binary data
stmt.setBytes(11, byteArray);
```

Insert with PreparedStatement:

```
public long insertProduct(Product product) {
    long newId = 0;
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;

    try {
        conn = DBConnectionPool.getConnection();

        // Create prepared statement with parameter placeholders
        stmt = conn.prepareStatement(
            "INSERT INTO products (name, description, price, category) VALUES (?,
?, ?, ?)",
            Statement.RETURN_GENERATED_KEYS
        );

        // Set parameters
        stmt.setString(1, product.getName());
        stmt.setString(2, product.getDescription());
        stmt.setDouble(3, product.getPrice());
        stmt.setString(4, product.getCategory());

        // Execute the update
        int rowsAffected = stmt.executeUpdate();

        // Get generated keys
        if (rowsAffected > 0) {
            rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                newId = rs.getLong(1);
                product.setId(newId);
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            if (conn != null) conn.close(); // Return to pool
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```
        return newId;
    }
```

Batch Processing with PreparedStatement:

```
public void insertProducts(List<Product> products) {
    Connection conn = null;
    PreparedStatement stmt = null;

    try {
        conn = DBConnectionPool.getConnection();

        // Create prepared statement
        stmt = conn.prepareStatement(
            "INSERT INTO products (name, description, price, category) VALUES (?,
?, ?, ?)"
        );

        // Add batches
        for (Product product : products) {
            stmt.setString(1, product.getName());
            stmt.setString(2, product.getDescription());
            stmt.setDouble(3, product.getPrice());
            stmt.setString(4, product.getCategory());

            // Add to batch
            stmt.addBatch();
        }

        // Execute batch
        int[] results = stmt.executeBatch();

        System.out.println("Batch insert completed. Records affected: " +
results.length);

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (stmt != null) stmt.close();
            if (conn != null) conn.close(); // Return to pool
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

CallableStatement

CallableStatement is used to execute SQL stored procedures and functions.

Benefits of CallableStatement:

1. **Code Reuse**:
    - Reuse database stored procedures
    - Centralize business logic in the database
2. **Performance**:
    - Stored procedures are precompiled
    - Reduced network traffic
    - Can be optimized by the database
3. **Security**:
    - Can restrict direct table access
    - Grant execute permissions on procedures
    - Parameter validation at the database level

Using CallableStatement for Stored Procedure:

```
public void updateProductPrice(long productId, double newPrice, String updatedBy)
{
    Connection conn = null;
    CallableStatement cstmt = null;

    try {
        conn = DBConnectionPool.getConnection();

        // Call stored procedure
        cstmt = conn.prepareCall("{call update_product_price(?, ?, ?)}");

        // Set parameters
        cstmt.setLong(1, productId);
        cstmt.setDouble(2, newPrice);
        cstmt.setString(3, updatedBy);

        // Execute procedure
        cstmt.execute();

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (cstmt != null) cstmt.close();
            if (conn != null) conn.close(); // Return to pool
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Using CallableStatement with OUT Parameters:

```
public double calculateOrderTotal(long orderId) {
    Connection conn = null;
    CallableStatement cstmt = null;
    double total = 0.0;

    try {
        conn = DBConnectionPool.getConnection();

        // Call stored procedure with OUT parameter
        cstmt = conn.prepareCall("{call calculate_order_total(?, ?)}");

        // Set IN parameter
        cstmt.setLong(1, orderId);

        // Register OUT parameter
        cstmt.registerOutParameter(2, java.sql.Types.DECIMAL);

        // Execute procedure
        cstmt.execute();

        // Get OUT parameter value
        total = cstmt.getDouble(2);

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (cstmt != null) cstmt.close();
            if (conn != null) conn.close(); // Return to pool
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    return total;
}
```

Using CallableStatement for Stored Function:

```
public boolean isProductInStock(long productId) {
    Connection conn = null;
    CallableStatement cstmt = null;
    boolean inStock = false;

    try {
        conn = DBConnectionPool.getConnection();

        // Call stored function
```

```
            cstmt = conn.prepareCall("{? = call is_product_in_stock(?)}");

            // Register return value
            cstmt.registerOutParameter(1, java.sql.Types.BOOLEAN);

            // Set parameter
            cstmt.setLong(2, productId);

            // Execute function
            cstmt.execute();

            // Get return value
            inStock = cstmt.getBoolean(1);

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (cstmt != null) cstmt.close();
                if (conn != null) conn.close(); // Return to pool
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return inStock;
    }
```

PreparedStatement vs CallableStatement vs Statement

Feature	Statement	PreparedStatement	CallableStatement
SQL Injection Protection	No	Yes	Yes
Performance for Multiple Executions	Poor	Good	Good
Database Caching	No	Yes	Yes
Parameter Handling	Manual	Automated	Automated
Stored Procedure Support	No	No	Yes
OUT Parameters	No	No	Yes
Batch Operations	Yes (limited)	Yes	Limited
Use Case	Simple, one-time queries	Parameterized queries	Stored procedures

Best Practices

1. **Always Use PreparedStatement Over Statement**:

- Better security
- Better performance
- Cleaner code

2. **Parameter Handling**:
   - Use type-specific setter methods
   - Handle null values properly
   - Use appropriate JDBC types
3. **Resource Management**:
   - Close statements when done
   - Use try-with-resources for automatic cleanup
   - Return connections to the pool
4. **Batch Processing**:
   - Use batching for bulk operations
   - Set appropriate batch sizes
   - Handle batch execution results
5. **Error Handling**:
   - Catch and log SQLException
   - Check SQL states for specific errors
   - Provide meaningful error messages
6. **Security Considerations**:
   - Don't construct SQL strings with concatenation
   - Use parameter binding for all user inputs
   - Validate input before database operations
7. **Performance Tuning**:
   - Reuse prepared statements when possible
   - Use connection pooling
   - Set appropriate fetch sizes for large result sets

4.5 Batch Processing

Batch processing allows executing multiple SQL statements in a single database roundtrip, significantly improving performance for bulk operations.

Benefits of Batch Processing

1. **Reduced Network Traffic**:
   - Fewer roundtrips between application and database
   - Less overhead for connection management

2. **Improved Performance**:
    - Faster execution of multiple operations
    - Database can optimize execution
3. **Transaction Efficiency**:
    - Single transaction for multiple operations
    - Reduced locking contention

Basic Batch Processing

Batch Insert Example:

```
public void batchInsertUsers(List<User> users) {
    Connection conn = null;
    PreparedStatement stmt = null;

    try {
        conn = DBConnectionPool.getConnection();
        conn.setAutoCommit(false); // Disable auto-commit for better performance

        stmt = conn.prepareStatement(
            "INSERT INTO users (username, email, department) VALUES (?, ?, ?)"
        );

        // Add statements to batch
        for (User user : users) {
            stmt.setString(1, user.getUsername());
            stmt.setString(2, user.getEmail());
            stmt.setString(3, user.getDepartment());
            stmt.addBatch();

            // Execute every 100 records for memory efficiency
            if (users.indexOf(user) % 100 == 0) {
                stmt.executeBatch();
                stmt.clearBatch();
            }
        }

        // Execute remaining records
        stmt.executeBatch();

        // Commit transaction
        conn.commit();

    } catch (SQLException e) {
        // Rollback on error
        try {
            if (conn != null) {
                conn.rollback();
            }
```

```
            } catch (SQLException ex) {
                ex.printStackTrace();
            }

            e.printStackTrace();
        } finally {
            try {
                if (stmt != null) stmt.close();
                if (conn != null) {
                    conn.setAutoCommit(true);
                    conn.close(); // Return to pool
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

Handling Batch Results

Processing Batch Execution Results:

```
    public void batchUpdateProducts(List<Product> products) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = DBConnectionPool.getConnection();
            conn.setAutoCommit(false);

            stmt = conn.prepareStatement(
                "UPDATE products SET price = ?, stock = ? WHERE id = ?"
            );

            // Add statements to batch
            for (Product product : products) {
                stmt.setDouble(1, product.getPrice());
                stmt.setInt(2, product.getStock());
                stmt.setLong(3, product.getId());
                stmt.addBatch();
            }

            // Execute batch and get results
            int[] updateCounts = stmt.executeBatch();

            // Process results
            int successCount = 0;
            int failureCount = 0;

            for (int i = 0; i < updateCounts.length; i++) {
                if (updateCounts[i] == Statement.SUCCESS_NO_INFO) {
```

```
                successCount++;
            } else if (updateCounts[i] == Statement.EXECUTE_FAILED) {
                failureCount++;
                System.out.println("Update failed for product: " +
products.get(i).getId());
            } else {
                // updateCounts[i] represents the number of rows affected
                successCount++;
            }
        }

        System.out.println("Batch update complete. Success: " + successCount + ",
Failures: " + failureCount);

        // Commit transaction
        conn.commit();

    } catch (BatchUpdateException e) {
        // Rollback on error
        try {
            if (conn != null) {
                conn.rollback();
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

        // Process partial results
        int[] updateCounts = e.getUpdateCounts();
        System.out.println("Batch update partially failed. Completed operations: "
+ updateCounts.length);

        e.printStackTrace();
    } catch (SQLException e) {
        // Rollback on error
        try {
            if (conn != null) {
                conn.rollback();
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

        e.printStackTrace();
    } finally {
        try {
            if (stmt != null) stmt.close();
            if (conn != null) {
                conn.setAutoCommit(true);
                conn.close(); // Return to pool
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

Multiple Statement Batching

Executing Different Types of Statements in Batch:

```
public void processOrder(Order order, List<OrderItem> items) {
    Connection conn = null;
    PreparedStatement orderStmt = null;
    PreparedStatement itemStmt = null;
    PreparedStatement updateStmt = null;
    ResultSet rs = null;

    try {
        conn = DBConnectionPool.getConnection();
        conn.setAutoCommit(false);

        // Insert order
        orderStmt = conn.prepareStatement(
            "INSERT INTO orders (user_id, order_date, status, total) VALUES (?, ?,
?, ?)",
            Statement.RETURN_GENERATED_KEYS
        );

        orderStmt.setLong(1, order.getUserId());
        orderStmt.setDate(2, new java.sql.Date(order.getOrderDate().getTime()));
        orderStmt.setString(3, order.getStatus());
        orderStmt.setDouble(4, order.getTotal());

        orderStmt.executeUpdate();

        // Get generated order ID
        rs = orderStmt.getGeneratedKeys();
        long orderId = 0;
        if (rs.next()) {
            orderId = rs.getLong(1);
            order.setId(orderId);
        } else {
            throw new SQLException("Creating order failed, no ID obtained.");
        }

        // Insert order items
        itemStmt = conn.prepareStatement(
            "INSERT INTO order_items (order_id, product_id, quantity, price)
VALUES (?, ?, ?, ?)"
        );

        for (OrderItem item : items) {
            itemStmt.setLong(1, orderId);
            itemStmt.setLong(2, item.getProductId());
```

```
                itemStmt.setInt(3, item.getQuantity());
                itemStmt.setDouble(4, item.getPrice());
                itemStmt.addBatch();
            }

            itemStmt.executeBatch();

            // Update product inventory
            updateStmt = conn.prepareStatement(
                "UPDATE products SET stock = stock - ? WHERE id = ?"
            );

            for (OrderItem item : items) {
                updateStmt.setInt(1, item.getQuantity());
                updateStmt.setLong(2, item.getProductId());
                updateStmt.addBatch();
            }

            updateStmt.executeBatch();

            // Commit transaction
            conn.commit();

        } catch (SQLException e) {
            // Rollback on error
            try {
                if (conn != null) {
                    conn.rollback();
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
            }

            e.printStackTrace();
            throw new RuntimeException("Order processing failed: " + e.getMessage(),
e);
        } finally {
            try {
                if (rs != null) rs.close();
                if (orderStmt != null) orderStmt.close();
                if (itemStmt != null) itemStmt.close();
                if (updateStmt != null) updateStmt.close();
                if (conn != null) {
                    conn.setAutoCommit(true);
                    conn.close(); // Return to pool
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

Rewritable Batch

Some JDBC drivers support the `rewriteBatchedStatements` connection parameter, which can dramatically improve batch insert performance:

```
// Connection URL with rewriteBatchedStatements for MySQL
String url = "jdbc:mysql://localhost:3306/mydb?rewriteBatchedStatements=true";
```

This causes the JDBC driver to rewrite multiple insert statements into a single multi-row insert:

```
-- Instead of multiple individual inserts:
INSERT INTO users (username, email) VALUES ('user1', 'user1@example.com');
INSERT INTO users (username, email) VALUES ('user2', 'user2@example.com');

-- The driver can rewrite as a single statement:
INSERT INTO users (username, email) VALUES
('user1', 'user1@example.com'),
('user2', 'user2@example.com');
```

CSV Import with Batch Processing

A common use case for batch processing is importing data from CSV files:

```
public void importUsersFromCsv(String csvFilePath) {
    Connection conn = null;
    PreparedStatement stmt = null;
    BufferedReader reader = null;
    String line;

    try {
        conn = DBConnectionPool.getConnection();
        conn.setAutoCommit(false);

        stmt = conn.prepareStatement(
            "INSERT INTO users (username, email, department) VALUES (?, ?, ?)"
        );

        reader = new BufferedReader(new FileReader(csvFilePath));

        // Skip header row
        reader.readLine();

        int count = 0;

        // Read and process each line
        while ((line = reader.readLine()) != null) {
            String[] data = line.split(",");
```

```
                if (data.length >= 3) {
                    stmt.setString(1, data[0].trim());
                    stmt.setString(2, data[1].trim());
                    stmt.setString(3, data[2].trim());
                    stmt.addBatch();

                    count++;

                    // Execute every 1000 rows
                    if (count % 1000 == 0) {
                        stmt.executeBatch();
                        stmt.clearBatch();
                        System.out.println("Processed " + count + " records");
                    }
                }
            }

            // Execute remaining batch
            if (count % 1000 != 0) {
                stmt.executeBatch();
            }

            // Commit transaction
            conn.commit();

            System.out.println("Import completed. Total records: " + count);

        } catch (Exception e) {
            // Rollback on error
            try {
                if (conn != null) {
                    conn.rollback();
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
            }

            e.printStackTrace();
        } finally {
            try {
                if (reader != null) reader.close();
                if (stmt != null) stmt.close();
                if (conn != null) {
                    conn.setAutoCommit(true);
                    conn.close(); // Return to pool
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

Batch Processing Best Practices

1. **Optimize Batch Size**:
    - Balance memory usage and performance
    - Typical batch sizes range from 100 to 1000
    - Test different batch sizes for your specific use case
2. **Use Transactions**:
    - Disable auto-commit for better performance
    - Explicitly commit after batch execution
    - Always provide rollback capability
3. **Error Handling**:
    - Use BatchUpdateException to handle partial failures
    - Implement retry logic for transient errors
    - Log failed operations for manual review
4. **Memory Management**:
    - Clear batches after execution
    - Process large datasets in chunks
    - Monitor memory usage
5. **Performance Considerations**:
    - Use database-specific optimizations (e.g., rewriteBatchedStatements)
    - Consider using JDBC driver batch optimizations
    - Minimize network roundtrips
6. **Monitoring and Debugging**:
    - Track batch execution times
    - Log batch sizes and success rates
    - Implement proper error reporting

5. Modern Integration Points

5.1 Servlets/JSP with Modern Frameworks

While Servlets and JSP provide a solid foundation, modern Java web development often uses frameworks that build upon them. Understanding how these technologies integrate is crucial for evolving legacy applications or building new ones.

Spring MVC Integration

Spring MVC is built on top of the Servlet API and provides a more structured approach to web development.

1. Basic Spring MVC Integration:

Step 1: Add Spring dependencies to pom.xml:

```
<dependencies>
    <!-- Servlet API -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>

    <!-- JSP API -->
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>2.3.3</version>
        <scope>provided</scope>
    </dependency>

    <!-- JSTL -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- Spring MVC -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.3.20</version>
    </dependency>
</dependencies>
```

Step 2: Configure DispatcherServlet in web.xml:

```
<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">

    <display-name>Spring MVC Application</display-name>

    <!-- Spring MVC Dispatcher Servlet -->
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
        <init-param>
```

```
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-mvc-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Step 3: Create Spring MVC configuration file (spring-mvc-config.xml):

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-
context.xsd
                           http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/mvc/spring-
mvc.xsd">

    <!-- Enable component scanning -->
    <context:component-scan base-package="com.example.controllers" />

    <!-- Enable annotation-driven Spring MVC -->
    <mvc:annotation-driven />

    <!-- Static resources -->
    <mvc:resources mapping="/resources/**" location="/resources/" />

    <!-- View resolver -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

Step 4: Create a Spring MVC controller:

```
package com.example.controllers;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class UserController {

    @GetMapping("/users")
    public String listUsers(Model model) {
        // Add data to the model
        model.addAttribute("message", "User list will be displayed here");

        // Return view name
        return "user/list";
    }

    @GetMapping("/users/add")
    public String showAddForm() {
        return "user/form";
    }

    @PostMapping("/users/add")
    public String addUser(@RequestParam("name") String name,
                          @RequestParam("email") String email,
                          Model model) {

        // Process form submission
        model.addAttribute("name", name);
        model.addAttribute("email", email);

        return "user/success";
    }
}
```

Step 5: Create JSP views:

/WEB-INF/views/user/list.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>User List</title>
    <link rel="stylesheet" href="<c:url value='/resources/css/style.css' />" />
</head>
<body>
    <h1>User List</h1>
```

```
    <p>${message}</p>

    <p><a href="<c:url value='/users/add' />">Add New User</a></p>
</body>
</html>
```

/WEB-INF/views/user/form.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Add User</title>
    <link rel="stylesheet" href="<c:url value='/resources/css/style.css' />" />
</head>
<body>
    <h1>Add User</h1>

    <form action="<c:url value='/users/add' />" method="post">
        <div>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>
        </div>
        <div>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
        </div>
        <div>
            <button type="submit">Save</button>
        </div>
    </form>
</body>
</html>
```

/WEB-INF/views/user/success.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Success</title>
    <link rel="stylesheet" href="<c:url value='/resources/css/style.css' />" />
</head>
```

```
<body>
    <h1>User Added Successfully</h1>

    <div>
        <p>Name: ${name}</p>
        <p>Email: ${email}</p>
    </div>

    <p><a href="<c:url value='/users' />">Back to User List</a></p>
</body>
</html>
```

2. Java Configuration (Alternative to XML):

Instead of XML configuration, you can use Java-based configuration:

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.example.controllers")
public class WebConfig implements WebMvcConfigurer {

    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver resolver = new
InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
        return resolver;
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**")
                .addResourceLocations("/resources/");
    }
}
```

And initialize it using a WebApplicationInitializer instead of web.xml:

```
package com.example.config;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import
org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class WebAppInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        // Create Spring web application context
        AnnotationConfigWebApplicationContext context = new
AnnotationConfigWebApplicationContext();
        context.register(WebConfig.class);

        // Create and register the DispatcherServlet
        DispatcherServlet servlet = new DispatcherServlet(context);
        ServletRegistration.Dynamic registration =
servletContext.addServlet("dispatcher", servlet);
        registration.setLoadOnStartup(1);
        registration.addMapping("/");
    }
}
```

3. Integrating Existing Servlets and JSPs with Spring MVC:

You can mix Spring MVC with traditional servlets:

```
// Traditional servlet
@WebServlet("/legacy/*")
public class LegacyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Servlet processing
        request.getRequestDispatcher("/WEB-INF/legacy-view.jsp").forward(request,
response);
    }
}

// Spring MVC controller
@Controller
public class ModernController {

    @GetMapping("/modern")
    public String handleRequest(Model model) {
```

```
        model.addAttribute("message", "This is handled by Spring MVC");
        return "modern-view";
    }
}
```

4. Accessing Spring Beans from Servlets:

```
@WebServlet("/hybrid-servlet")
public class HybridServlet extends HttpServlet {

    private UserService userService;

    @Override
    public void init() throws ServletException {
        // Get Spring web application context
        WebApplicationContext context = WebApplicationContextUtils
            .getWebApplicationContext(getServletContext());

        // Get bean from Spring context
        userService = context.getBean(UserService.class);
    }

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

        // Use Spring bean
        List<User> users = userService.getAllUsers();

        // Set attribute and forward to JSP
        request.setAttribute("users", users);
        request.getRequestDispatcher("/WEB-INF/views/hybrid-
view.jsp").forward(request, response);
    }
}
```

Jakarta EE Integration

Jakarta EE (formerly Java EE) provides its own MVC framework and other enterprise features that build on the Servlet/JSP foundation.

1. CDI (Contexts and Dependency Injection) with Servlets:

```
import javax.inject.Inject;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet("/cdi-servlet")
public class CDIServlet extends HttpServlet {
```

```
    @Inject
    private UserService userService;

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

        List<User> users = userService.getAllUsers();
        request.setAttribute("users", users);
        request.getRequestDispatcher("/WEB-INF/views/users.jsp").forward(request,
response);
    }
}
```

2. JPA (Java Persistence API) Integration:

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.servlet.annotation.WebServlet;
import javax.transaction.Transactional;

@WebServlet("/jpa-servlet")
public class JPAServlet extends HttpServlet {

    @PersistenceContext
    private EntityManager entityManager;

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

        List<User> users = entityManager.createQuery("SELECT u FROM User u",
User.class)
                                        .getResultList();

        request.setAttribute("users", users);
        request.getRequestDispatcher("/WEB-INF/views/users.jsp").forward(request,
response);
    }

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

        String name = request.getParameter("name");
        String email = request.getParameter("email");

        User user = new User();
        user.setName(name);
```

```
        user.setEmail(email);

        entityManager.persist(user);

        response.sendRedirect(request.getContextPath() + "/jpa-servlet");
    }
}
```

3. Bean Validation Integration:

```
import javax.inject.Inject;
import javax.servlet.annotation.WebServlet;
import javax.validation.ConstraintViolation;
import javax.validation.Validator;
import java.util.Set;

@WebServlet("/validation-servlet")
public class ValidationServlet extends HttpServlet {

    @Inject
    private Validator validator;

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

        String name = request.getParameter("name");
        String email = request.getParameter("email");

        User user = new User();
        user.setName(name);
        user.setEmail(email);

        // Validate user
        Set<ConstraintViolation<User>> violations = validator.validate(user);

        if (violations.isEmpty()) {
            // Validation passed
            // Process the valid user...
            response.sendRedirect(request.getContextPath() + "/success.jsp");
        } else {
            // Validation failed
            request.setAttribute("violations", violations);
            request.setAttribute("user", user);
            request.getRequestDispatcher("/WEB-
INF/views/form.jsp").forward(request, response);
        }
    }
}
```

User Model with Bean Validation:

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;

@Entity
public class User {

    @Id
    @GeneratedValue
    private Long id;

    @NotBlank(message = "Name is required")
    @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")
    private String name;

    @NotBlank(message = "Email is required")
    @Email(message = "Email must be valid")
    private String email;

    // Getters and setters
}
```

JAX-RS (REST API) Integration

JAX-RS provides a framework for creating RESTful web services and can integrate with servlets.

1. JAX-RS REST Service:

```
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import java.util.List;

@Path("/api/users")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class UserResource {

    private UserService userService = new UserService();

    @GET
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @GET
```

```
    @Path("/{id}")
    public Response getUserById(@PathParam("id") long id) {
        User user = userService.getUserById(id);
        if (user == null) {
            return Response.status(Response.Status.NOT_FOUND).build();
        }
        return Response.ok(user).build();
    }

    @POST
    public Response createUser(User user) {
        userService.addUser(user);
        return Response.status(Response.Status.CREATED)
                       .entity(user)
                       .build();
    }

    @PUT
    @Path("/{id}")
    public Response updateUser(@PathParam("id") long id, User user) {
        user.setId(id);
        userService.updateUser(user);
        return Response.ok(user).build();
    }

    @DELETE
    @Path("/{id}")
    public Response deleteUser(@PathParam("id") long id) {
        userService.deleteUser(id);
        return Response.noContent().build();
    }
}
```

2. Configuring JAX-RS in web.xml:

```
<servlet>
    <servlet-name>jersey-servlet</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.example.resources</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>jersey-servlet</servlet-name>
    <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

3. Java Client for JAX-RS Services:

```
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.core.GenericType;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.util.List;

public class UserClient {

    private static final String BASE_URI = "http://localhost:8080/api/users";
    private Client client;

    public UserClient() {
        client = ClientBuilder.newClient();
    }

    public List<User> getAllUsers() {
        return client.target(BASE_URI)
                     .request(MediaType.APPLICATION_JSON)
                     .get(new GenericType<List<User>>(){});
    }

    public User getUserById(long id) {
        return client.target(BASE_URI)
                     .path(String.valueOf(id))
                     .request(MediaType.APPLICATION_JSON)
                     .get(User.class);
    }

    public User createUser(User user) {
        Response response = client.target(BASE_URI)
                                  .request(MediaType.APPLICATION_JSON)
                                  .post(Entity.entity(user,
MediaType.APPLICATION_JSON));

        return response.readEntity(User.class);
    }

    public User updateUser(long id, User user) {
        Response response = client.target(BASE_URI)
                                  .path(String.valueOf(id))
                                  .request(MediaType.APPLICATION_JSON)
                                  .put(Entity.entity(user,
MediaType.APPLICATION_JSON));

        return response.readEntity(User.class);
    }

    public void deleteUser(long id) {
        client.target(BASE_URI)
              .path(String.valueOf(id))
              .request()
```

```
                .delete();
    }

    public void close() {
        if (client != null) {
            client.close();
        }
    }
}
```

4. Using a JAX-RS Client in a Servlet:

```
@WebServlet("/user-proxy")
public class UserProxyServlet extends HttpServlet {

    private UserClient userClient;

    @Override
    public void init() throws ServletException {
        userClient = new UserClient();
    }

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

        String action = request.getParameter("action");

        if ("list".equals(action)) {
            // Get all users from the REST service
            List<User> users = userClient.getAllUsers();
            request.setAttribute("users", users);
            request.getRequestDispatcher("/WEB-INF/views/user-
list.jsp").forward(request, response);

        } else if ("view".equals(action)) {
            // Get specific user by ID
            long id = Long.parseLong(request.getParameter("id"));
            User user = userClient.getUserById(id);
            request.setAttribute("user", user);
            request.getRequestDispatcher("/WEB-INF/views/user-
detail.jsp").forward(request, response);

        } else {
            response.sendRedirect(request.getContextPath() + "/user-proxy?
action=list");
        }
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
```

```
            throws ServletException, IOException {

        String action = request.getParameter("action");

        if ("create".equals(action)) {
            // Create new user
            User user = new User();
            user.setName(request.getParameter("name"));
            user.setEmail(request.getParameter("email"));

            userClient.createUser(user);
            response.sendRedirect(request.getContextPath() + "/user-proxy?
action=list");

        } else if ("update".equals(action)) {
            // Update existing user
            long id = Long.parseLong(request.getParameter("id"));
            User user = new User();
            user.setName(request.getParameter("name"));
            user.setEmail(request.getParameter("email"));

            userClient.updateUser(id, user);
            response.sendRedirect(request.getContextPath() + "/user-proxy?
action=view&id=" + id);

        } else if ("delete".equals(action)) {
            // Delete user
            long id = Long.parseLong(request.getParameter("id"));
            userClient.deleteUser(id);
            response.sendRedirect(request.getContextPath() + "/user-proxy?
action=list");
        }
    }

    @Override
    public void destroy() {
        if (userClient != null) {
            userClient.close();
        }
    }
}
```

Thymeleaf Integration

Thymeleaf is a modern server-side Java template engine that can replace JSP while working with servlets.

1. Setting Up Thymeleaf with Servlets:

Add Thymeleaf dependencies:

```
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf</artifactId>
    <version>3.0.15.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-java8time</artifactId>
    <version>3.0.4.RELEASE</version>
</dependency>
```

Create a Thymeleaf configuration class:

```
package com.example.config;

import org.thymeleaf.TemplateEngine;
import org.thymeleaf.templatemode.TemplateMode;
import org.thymeleaf.templateresolver.ServletContextTemplateResolver;

import javax.servlet.ServletContext;

public class ThymeleafConfig {

    private static TemplateEngine templateEngine;

    public static void initialize(ServletContext servletContext) {
        // Create template resolver
        ServletContextTemplateResolver templateResolver = new
ServletContextTemplateResolver(servletContext);
        templateResolver.setTemplateMode(TemplateMode.HTML);
        templateResolver.setPrefix("/WEB-INF/templates/");
        templateResolver.setSuffix(".html");
        templateResolver.setCacheTTLMs(3600000L); // TTL=1h
        templateResolver.setCacheable(true);

        // Create template engine
        templateEngine = new TemplateEngine();
        templateEngine.setTemplateResolver(templateResolver);
        templateEngine.addDialect(new
org.thymeleaf.extras.java8time.dialect.Java8TimeDialect());
    }

    public static TemplateEngine getTemplateEngine() {
        return templateEngine;
    }
}
```

Initialize Thymeleaf in a ServletContextListener:

```
@WebListener
public class ThymeleafInitializer implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ThymeleafConfig.initialize(sce.getServletContext());
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        // Cleanup if needed
    }
}
```

2. Using Thymeleaf in a Servlet:

```
@WebServlet("/thymeleaf-demo")
public class ThymeleafServlet extends HttpServlet {

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

        // Set model data
        List<User> users = getUserList(); // Sample method to get users

        // Prepare the context
        WebContext context = new WebContext(request, response,
getServletContext());
        context.setVariable("users", users);
        context.setVariable("appName", "Thymeleaf Demo");
        context.setVariable("currentDate", LocalDate.now());

        // Process the template
        TemplateEngine templateEngine = ThymeleafConfig.getTemplateEngine();
        templateEngine.process("users", context, response.getWriter());
    }

    private List<User> getUserList() {
        // Sample data for demonstration
        List<User> users = new ArrayList<>();
        users.add(new User(1L, "John Doe", "john@example.com"));
        users.add(new User(2L, "Jane Smith", "jane@example.com"));
        users.add(new User(3L, "Bob Johnson", "bob@example.com"));
        return users;
    }
}
```

3. Thymeleaf Template Example (users.html):

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title th:text="${appName}">App Name</title>
    <style>
      table {
        border-collapse: collapse;
        width: 100%;
      }
      th,
      td {
        border: 1px solid #ddd;
        padding: 8px;
        text-align: left;
      }
      th {
        background-color: #f2f2f2;
      }
    </style>
  </head>
  <body>
    <h1 th:text="${appName}">App Name</h1>
    <p>
      Today is:
      <span th:text="${#temporals.format(currentDate, 'dd MMMM yyyy')}"
        >1 January 2023</span
      >
    </p>

    <h2>User List</h2>

    <table>
      <thead>
        <tr>
          <th>ID</th>
          <th>Name</th>
          <th>Email</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="user : ${users}">
          <td th:text="${user.id}">1</td>
          <td th:text="${user.name}">John Doe</td>
          <td th:text="${user.email}">john@example.com</td>
          <td>
            <a th:href="@{/thymeleaf-demo/edit(id=${user.id})}">Edit</a> |
            <a
              th:href="@{/thymeleaf-demo/delete(id=${user.id})}"
              onclick="return confirm('Are you sure?')"
              >Delete</a
            >
```

```
            </td>
          </tr>
          <tr th:if="${users.empty}">
            <td colspan="4">No users found</td>
          </tr>
        </tbody>
      </table>

      <p><a th:href="@{/thymeleaf-demo/add}">Add New User</a></p>
    </body>
  </html>
```

Best Practices for Modern Integration

1. **Choose the Right Technology for the Job**:
    - Use Spring MVC for complex applications with extensive middleware needs
    - Use JAX-RS for RESTful APIs
    - Consider Thymeleaf or other template engines instead of JSP for new projects
2. **Incremental Migration Strategy**:
    - Start by introducing frameworks alongside existing servlets
    - Gradually migrate functionality to modern frameworks
    - Use dependency injection to simplify integration
3. **Leverage Common Infrastructure**:
    - Use shared data sources and connection pools
    - Implement common security mechanisms
    - Share service layer components
4. **Consistent Error Handling**:
    - Implement unified error handling across technologies
    - Create consistent error pages
    - Use common logging strategies
5. **Testing Considerations**:
    - Test integrated components together
    - Use integration tests to ensure proper interaction
    - Test with different browsers and client types
6. **Documentation**:
    - Document integration points thoroughly
    - Create architecture diagrams showing relationships
    - Document migration paths for future development

5.2 REST API Development with Servlets

Building RESTful APIs directly with servlets provides fine-grained control over HTTP operations and response formats.

Basic REST Principles

1. **Resources**: Everything is a resource (users, products, orders)
2. **URIs**: Resources are identified by URIs (/users, /products/123)
3. **HTTP Methods**: Use standard HTTP methods (GET, POST, PUT, DELETE)
4. **Representations**: Resources can have multiple representations (JSON, XML)
5. **Stateless**: Each request contains all information needed to process it

Building a REST API with Servlets

1. Basic Structure:

```
@WebServlet("/api/users/*")
public class UserRestServlet extends HttpServlet {

    private UserService userService = new UserService();
    private ObjectMapper objectMapper = new ObjectMapper(); // Jackson JSON
processor

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

        // Set response type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        // Get the path info
        String pathInfo = request.getPathInfo();

        if (pathInfo == null || pathInfo.equals("/")) {
            // Get all users
            List<User> users = userService.getAllUsers();
            objectMapper.writeValue(response.getWriter(), users);

        } else {
            try {
                // Get user by ID
                long id = Long.parseLong(pathInfo.substring(1));
                User user = userService.getUserById(id);

                if (user != null) {
                    objectMapper.writeValue(response.getWriter(), user);
                } else {
                    response.setStatus(HttpServletResponse.SC_NOT_FOUND);
                    Map<String, String> error = new HashMap<>();
                    error.put("error", "User not found");
                    objectMapper.writeValue(response.getWriter(), error);
```

```
                }

            } catch (NumberFormatException e) {
                response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                Map<String, String> error = new HashMap<>();
                error.put("error", "Invalid user ID format");
                objectMapper.writeValue(response.getWriter(), error);
            }
        }
    }

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

        // Set response type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        try {
            // Parse user from request body
            User user = objectMapper.readValue(request.getInputStream(),
User.class);

            // Validate user
            if (user.getName() == null || user.getName().trim().isEmpty()) {
                response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                Map<String, String> error = new HashMap<>();
                error.put("error", "Name is required");
                objectMapper.writeValue(response.getWriter(), error);
                return;
            }

            // Save user
            userService.addUser(user);

            // Return created user with status 201
            response.setStatus(HttpServletResponse.SC_CREATED);
            objectMapper.writeValue(response.getWriter(), user);

        } catch (Exception e) {
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            Map<String, String> error = new HashMap<>();
            error.put("error", "Failed to create user: " + e.getMessage());
            objectMapper.writeValue(response.getWriter(), error);
        }
    }

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

        // Set response type
```

```
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        // Get the path info for ID
        String pathInfo = request.getPathInfo();

        if (pathInfo == null || pathInfo.equals("/")) {
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            Map<String, String> error = new HashMap<>();
            error.put("error", "User ID is required");
            objectMapper.writeValue(response.getWriter(), error);
            return;
        }

        try {
            // Get user ID from path
            long id = Long.parseLong(pathInfo.substring(1));

            // Check if user exists
            User existingUser = userService.getUserById(id);
            if (existingUser == null) {
                response.setStatus(HttpServletResponse.SC_NOT_FOUND);
                Map<String, String> error = new HashMap<>();
                error.put("error", "User not found");
                objectMapper.writeValue(response.getWriter(), error);
                return;
            }

            // Parse updated user from request body
            User updatedUser = objectMapper.readValue(request.getInputStream(),
User.class);
            updatedUser.setId(id); // Ensure ID is set correctly

            // Update user
            userService.updateUser(updatedUser);

            // Return updated user
            objectMapper.writeValue(response.getWriter(), updatedUser);

        } catch (NumberFormatException e) {
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            Map<String, String> error = new HashMap<>();
            error.put("error", "Invalid user ID format");
            objectMapper.writeValue(response.getWriter(), error);
        } catch (Exception e) {
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            Map<String, String> error = new HashMap<>();
            error.put("error", "Failed to update user: " + e.getMessage());
            objectMapper.writeValue(response.getWriter(), error);
        }
    }

    @Override
    protected void doDelete(HttpServletRequest request, HttpServletResponse
```

```
response)
            throws ServletException, IOException {

        // Set response type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        // Get the path info for ID
        String pathInfo = request.getPathInfo();

        if (pathInfo == null || pathInfo.equals("/")) {
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            Map<String, String> error = new HashMap<>();
            error.put("error", "User ID is required");
            objectMapper.writeValue(response.getWriter(), error);
            return;
        }

        try {
            // Get user ID from path
            long id = Long.parseLong(pathInfo.substring(1));

            // Check if user exists
            User existingUser = userService.getUserById(id);
            if (existingUser == null) {
                response.setStatus(HttpServletResponse.SC_NOT_FOUND);
                Map<String, String> error = new HashMap<>();
                error.put("error", "User not found");
                objectMapper.writeValue(response.getWriter(), error);
                return;
            }

            // Delete user
            userService.deleteUser(id);

            // Return 204 No Content
            response.setStatus(HttpServletResponse.SC_NO_CONTENT);

        } catch (NumberFormatException e) {
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            Map<String, String> error = new HashMap<>();
            error.put("error", "Invalid user ID format");
            objectMapper.writeValue(response.getWriter(), error);
        } catch (Exception e) {
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            Map<String, String> error = new HashMap<>();
            error.put("error", "Failed to delete user: " + e.getMessage());
            objectMapper.writeValue(response.getWriter(), error);
        }
    }
}
```

2. Handling Content Negotiation:

```
@WebServlet("/api/products/*")
public class ProductRestServlet extends HttpServlet {

    private ProductService productService = new ProductService();
    private ObjectMapper jsonMapper = new ObjectMapper(); // Jackson JSON
processor
    private XmlMapper xmlMapper = new XmlMapper(); // Jackson XML processor

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

        // Check Accept header for content negotiation
        String acceptHeader = request.getHeader("Accept");
        boolean wantsXml = acceptHeader != null &&
acceptHeader.contains("application/xml");

        // Set response content type
        if (wantsXml) {
            response.setContentType("application/xml");
        } else {
            response.setContentType("application/json");
        }
        response.setCharacterEncoding("UTF-8");

        // Get the path info
        String pathInfo = request.getPathInfo();

        if (pathInfo == null || pathInfo.equals("/")) {
            // Get all products
            List<Product> products = productService.getAllProducts();

            if (wantsXml) {
                xmlMapper.writeValue(response.getWriter(), products);
            } else {
                jsonMapper.writeValue(response.getWriter(), products);
            }
        } else {
            try {
                // Get product by ID
                long id = Long.parseLong(pathInfo.substring(1));
                Product product = productService.getProductById(id);

                if (product != null) {
                    if (wantsXml) {
                        xmlMapper.writeValue(response.getWriter(), product);
                    } else {
                        jsonMapper.writeValue(response.getWriter(), product);
                    }
                } else {
                    response.setStatus(HttpServletResponse.SC_NOT_FOUND);
                    Map<String, String> error = new HashMap<>();
```

```
                    error.put("error", "Product not found");

                    if (wantsXml) {
                        xmlMapper.writeValue(response.getWriter(), error);
                    } else {
                        jsonMapper.writeValue(response.getWriter(), error);
                    }
                }
            } catch (NumberFormatException e) {
                response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                Map<String, String> error = new HashMap<>();
                error.put("error", "Invalid product ID format");

                if (wantsXml) {
                    xmlMapper.writeValue(response.getWriter(), error);
                } else {
                    jsonMapper.writeValue(response.getWriter(), error);
                }
            }
        }
    }

    // Implement other methods (POST, PUT, DELETE) with content negotiation...
}
```

3. Supporting CORS (Cross-Origin Resource Sharing):

```
@WebFilter("/api/*")
public class CorsFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Add CORS headers
        httpResponse.setHeader("Access-Control-Allow-Origin", "*");
        httpResponse.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT,
DELETE, OPTIONS");
        httpResponse.setHeader("Access-Control-Allow-Headers", "Content-Type,
Authorization");
        httpResponse.setHeader("Access-Control-Max-Age", "3600");

        // Handle preflight requests
        if (httpRequest.getMethod().equals("OPTIONS")) {
            httpResponse.setStatus(HttpServletResponse.SC_OK);
            return;
        }
```

```
        // Pass request down the filter chain
        chain.doFilter(request, response);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

4. Supporting Pagination:

```
@WebServlet("/api/products")
public class ProductPaginationServlet extends HttpServlet {

    private ProductService productService = new ProductService();
    private ObjectMapper objectMapper = new ObjectMapper();

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

        // Set response type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        try {
            // Parse pagination parameters
            int page = 1;
            int pageSize = 10;

            String pageParam = request.getParameter("page");
            if (pageParam != null && !pageParam.isEmpty()) {
                page = Integer.parseInt(pageParam);
            }

            String pageSizeParam = request.getParameter("pageSize");
            if (pageSizeParam != null && !pageSizeParam.isEmpty()) {
                pageSize = Integer.parseInt(pageSizeParam);
            }

            // Validate pagination parameters
            if (page < 1 || pageSize < 1 || pageSize > 100) {
                response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                Map<String, String> error = new HashMap<>();
                error.put("error", "Invalid pagination parameters. Page must be >=
1 and pageSize must be between 1-100");
```

```
                objectMapper.writeValue(response.getWriter(), error);
                return;
            }

            // Parse filter parameters
            String category = request.getParameter("category");
            String minPriceParam = request.getParameter("minPrice");
            String maxPriceParam = request.getParameter("maxPrice");

            Double minPrice = null;
            Double maxPrice = null;

            if (minPriceParam != null && !minPriceParam.isEmpty()) {
                minPrice = Double.parseDouble(minPriceParam);
            }

            if (maxPriceParam != null && !maxPriceParam.isEmpty()) {
                maxPrice = Double.parseDouble(maxPriceParam);
            }

            // Get paginated and filtered result
            int offset = (page - 1) * pageSize;
            PaginatedResult<Product> result = productService.getProducts(
                    offset, pageSize, category, minPrice, maxPrice);

            // Create response with pagination metadata
            Map<String, Object> response = new HashMap<>();
            response.put("data", result.getItems());
            response.put("pagination", Map.of(
                "page", page,
                "pageSize", pageSize,
                "totalItems", result.getTotalCount(),
                "totalPages", (int) Math.ceil((double) result.getTotalCount() /
pageSize)
            ));

            objectMapper.writeValue(response.getWriter(), response);

        } catch (NumberFormatException e) {
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            Map<String, String> error = new HashMap<>();
            error.put("error", "Invalid numerical parameter format");
            objectMapper.writeValue(response.getWriter(), error);
        } catch (Exception e) {
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            Map<String, String> error = new HashMap<>();
            error.put("error", "An error occurred: " + e.getMessage());
            objectMapper.writeValue(response.getWriter(), error);
        }
    }
}
```

Paginated Result Model:

```
public class PaginatedResult<T> {
    private List<T> items;
    private long totalCount;

    public PaginatedResult(List<T> items, long totalCount) {
        this.items = items;
        this.totalCount = totalCount;
    }

    public List<T> getItems() {
        return items;
    }

    public long getTotalCount() {
        return totalCount;
    }
}
```

5. Authentication and Authorization for REST APIs:

```
@WebFilter("/api/*")
public class AuthenticationFilter implements Filter {

    private AuthService authService = new AuthService();

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Skip authentication for OPTIONS requests (CORS preflight)
        if (httpRequest.getMethod().equals("OPTIONS")) {
            chain.doFilter(request, response);
            return;
        }

        // Get authorization header
        String authHeader = httpRequest.getHeader("Authorization");

        // Check if auth header is present and has correct format
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            httpResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            httpResponse.setContentType("application/json");
            httpResponse.getWriter().write("{\"error\":\"Missing or invalid
authorization\"}");
            return;
        }
```

```
        // Extract token
        String token = authHeader.substring(7); // Remove "Bearer " prefix

        try {
            // Validate token
            UserDetails userDetails = authService.validateToken(token);

            // Store user details in request for later use
            httpRequest.setAttribute("userDetails", userDetails);

            // Check role-based access
            if (isRestrictedEndpoint(httpRequest) && !hasAdminRole(userDetails)) {
                httpResponse.setStatus(HttpServletResponse.SC_FORBIDDEN);
                httpResponse.setContentType("application/json");
                httpResponse.getWriter().write("{\"error\":\"Insufficient
permissions\"}");
                return;
            }

            // Proceed with the request
            chain.doFilter(request, response);

        } catch (InvalidTokenException e) {
            httpResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            httpResponse.setContentType("application/json");
            httpResponse.getWriter().write("{\"error\":\"" + e.getMessage() +
"\"}");
        } catch (Exception e) {
            httpResponse.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            httpResponse.setContentType("application/json");
            httpResponse.getWriter().write("{\"error\":\"Authentication
error\"}");
        }
    }

    private boolean isRestrictedEndpoint(HttpServletRequest request) {
        String path = request.getRequestURI();
        String method = request.getMethod();

        // Check if endpoint requires admin access
        return (path.contains("/api/admin/") ||
                (path.contains("/api/users") && (method.equals("DELETE") ||
method.equals("PUT"))));
    }

    private boolean hasAdminRole(UserDetails userDetails) {
        return userDetails.getRoles().contains("ADMIN");
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }
```

```
    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

REST API Documentation with Swagger

Adding Swagger documentation to your servlet-based REST API:

1. Add Swagger Dependencies:

```
<dependency>
    <groupId>io.swagger.core.v3</groupId>
    <artifactId>swagger-jaxrs2</artifactId>
    <version>2.1.13</version>
</dependency>
<dependency>
    <groupId>io.swagger.core.v3</groupId>
    <artifactId>swagger-jaxrs2-servlet-initializer</artifactId>
    <version>2.1.13</version>
</dependency>
```

2. Create Swagger Configuration:

```
@WebServlet("/api/swagger.json")
public class SwaggerServlet extends HttpServlet {

    private ObjectMapper mapper = new ObjectMapper();

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

        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        // Create OpenAPI instance
        OpenAPI openAPI = new OpenAPI()
            .info(new Info()
                .title("User Management API")
                .description("API for managing users")
                .version("1.0.0")
                .contact(new Contact()
                    .name("API Support")
                    .email("support@example.com")))
            .servers(List.of(
                new Server().url("http://localhost:8080/api")))
```

```
            .paths(createPaths());

        // Write OpenAPI to response
        mapper.writeValue(response.getWriter(), openAPI);
    }

    private Paths createPaths() {
        Paths paths = new Paths();

        // /users path
        PathItem usersPath = new PathItem()
            .get(new Operation()
                .summary("Get all users")
                .description("Returns a list of all users")
                .responses(new ApiResponses()
                    .addApiResponse("200", new ApiResponse()
                        .description("Successful operation")
                        .content(new Content()
                            .addMediaType("application/json", new MediaType()
                                .schema(new ArraySchema()
                                    .items(createUserSchema()))))))
            .post(new Operation()
                .summary("Create a new user")
                .description("Creates a new user with the provided data")
                .requestBody(new RequestBody()
                    .content(new Content()
                        .addMediaType("application/json", new MediaType()
                            .schema(createUserSchema()))))
                .responses(new ApiResponses()
                    .addApiResponse("201", new ApiResponse()
                        .description("User created successfully")
                        .content(new Content()
                            .addMediaType("application/json", new MediaType()
                                .schema(createUserSchema()))))
                    .addApiResponse("400", new ApiResponse()
                        .description("Invalid input"))));

        // /users/{id} path
        PathItem userPath = new PathItem()
            .get(new Operation()
                .summary("Get user by ID")
                .description("Returns a single user by ID")
                .parameters(List.of(
                    new Parameter()
                        .name("id")
                        .in("path")
                        .required(true)
                        .schema(new Schema<>().type("integer").format("int64"))))
                .responses(new ApiResponses()
                    .addApiResponse("200", new ApiResponse()
                        .description("Successful operation")
                        .content(new Content()
                            .addMediaType("application/json", new MediaType()
                                .schema(createUserSchema()))))
```

```
                    .addApiResponse("404", new ApiResponse()
                        .description("User not found"))))
            .put(new Operation()
                .summary("Update user")
                .description("Updates an existing user")
                .parameters(List.of(
                    new Parameter()
                        .name("id")
                        .in("path")
                        .required(true)
                        .schema(new Schema<>().type("integer").format("int64"))))
                .requestBody(new RequestBody()
                    .content(new Content()
                        .addMediaType("application/json", new MediaType()
                            .schema(createUserSchema()))))
                .responses(new ApiResponses()
                    .addApiResponse("200", new ApiResponse()
                        .description("User updated successfully"))
                    .addApiResponse("404", new ApiResponse()
                        .description("User not found"))
                    .addApiResponse("400", new ApiResponse()
                        .description("Invalid input"))))
            .delete(new Operation()
                .summary("Delete user")
                .description("Deletes a user by ID")
                .parameters(List.of(
                    new Parameter()
                        .name("id")
                        .in("path")
                        .required(true)
                        .schema(new Schema<>().type("integer").format("int64"))))
                .responses(new ApiResponses()
                    .addApiResponse("204", new ApiResponse()
                        .description("User deleted successfully"))
                    .addApiResponse("404", new ApiResponse()
                        .description("User not found"))));

        paths.addPathItem("/users", usersPath);
        paths.addPathItem("/users/{id}", userPath);

        return paths;
    }

    private Schema createUserSchema() {
        return new Schema<>()
            .type("object")
            .properties(Map.of(
                "id", new Schema<>().type("integer").format("int64"),
                "name", new Schema<>().type("string"),
                "email", new Schema<>().type("string"),
                "role", new Schema<>().type("string").enum_(List.of("USER",
"ADMIN"))
            ))
            .required(List.of("name", "email"));
```

```
    }
}
```

3. Serve Swagger UI:

```
@WebServlet("/api-docs/*")
public class SwaggerUIServlet extends HttpServlet {

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

        String pathInfo = request.getPathInfo();

        if (pathInfo == null || pathInfo.equals("/")) {
            // Serve Swagger UI HTML
            response.setContentType("text/html");
            response.setCharacterEncoding("UTF-8");

            try (PrintWriter out = response.getWriter()) {
                out.println("<!DOCTYPE html>");
                out.println("<html>");
                out.println("<head>");
                out.println("  <title>API Documentation</title>");
                out.println("  <link rel=\"stylesheet\" type=\"text/css\"
href=\"https://unpkg.com/swagger-ui-dist@4.5.0/swagger-ui.css\" />");
                out.println("</head>");
                out.println("<body>");
                out.println("  <div id=\"swagger-ui\"></div>");
                out.println("  <script src=\"https://unpkg.com/swagger-ui-
dist@4.5.0/swagger-ui-bundle.js\"></script>");
                out.println("  <script>");
                out.println("    window.onload = function() {");
                out.println("      SwaggerUIBundle({");
                out.println("        url: \"" + request.getContextPath() +
"/api/swagger.json\",");
                out.println("        dom_id: '#swagger-ui',");
                out.println("        presets: [");
                out.println("          SwaggerUIBundle.presets.apis,");
                out.println("
SwaggerUIBundle.SwaggerUIStandalonePreset");
                out.println("        ],");
                out.println("        layout: \"BaseLayout\"");
                out.println("      });");
                out.println("    };");
                out.println("  </script>");
                out.println("</body>");
                out.println("</html>");
            }
        } else {
            response.sendError(HttpServletResponse.SC_NOT_FOUND);
        }
```

```
        }
    }
```

REST API Best Practices

1. **Use Proper HTTP Methods**:
    - GET for reading resources
    - POST for creating resources
    - PUT for updating resources
    - DELETE for removing resources
    - PATCH for partial updates
2. **Use Correct Status Codes**:
    - 200 OK for successful operations
    - 201 Created for resource creation
    - 204 No Content for successful deletion
    - 400 Bad Request for invalid input
    - 401 Unauthorized for authentication failures
    - 403 Forbidden for authorization failures
    - 404 Not Found for missing resources
    - 500 Internal Server Error for server issues
3. **Design Consistent URLs**:
    - Use plural nouns for collections (/users)
    - Use singular nouns with IDs for specific resources (/users/123)
    - Use nested resources for relationships (/users/123/orders)
    - Keep URLs simple and intuitive
4. **Implement Content Negotiation**:
    - Support multiple formats (JSON, XML)
    - Use Accept headers
    - Provide consistent error formats
5. **Versioning**:
    - Include version in URL (/api/v1/users)
    - Use Accept headers for versioning
6. **Pagination and Filtering**:
    - Support pagination parameters
    - Allow filtering by relevant attributes
    - Include pagination metadata in responses
7. **Security Considerations**:

- Implement proper authentication
- Use HTTPS
- Validate all inputs
- Implement CORS properly
- Rate limiting for API protection

8. **Documentation**:

- Document all endpoints
- Include example requests and responses
- Describe error conditions
- Specify required headers

5.3 JSON Processing

JSON (JavaScript Object Notation) is a lightweight data interchange format widely used in modern web applications. Java provides several libraries for processing JSON data.

Using Jackson for JSON Processing

Jackson is a popular Java library for working with JSON.

1. Adding Jackson Dependencies:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.4</version>
</dependency>
```

2. Basic JSON Serialization (Java Object to JSON):

```
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonSerializationExample {

    public static void main(String[] args) {
        try {
            // Create an ObjectMapper
            ObjectMapper mapper = new ObjectMapper();

            // Create a user object
            User user = new User();
            user.setId(1L);
            user.setName("John Doe");
            user.setEmail("john@example.com");
            user.setRoles(List.of("USER", "EDITOR"));

            // Serialize to JSON string
```

```
            String jsonString = mapper.writeValueAsString(user);
            System.out.println(jsonString);

            // Serialize to file
            mapper.writeValue(new File("user.json"), user);

            // Serialize to byte array
            byte[] jsonBytes = mapper.writeValueAsBytes(user);

            // Pretty print JSON
            String prettyJson =
mapper.writerWithDefaultPrettyPrinter().writeValueAsString(user);
            System.out.println(prettyJson);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. Basic JSON Deserialization (JSON to Java Object):

```
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonDeserializationExample {

    public static void main(String[] args) {
        try {
            // Create an ObjectMapper
            ObjectMapper mapper = new ObjectMapper();

            // JSON string to deserialize
            String jsonString = "{\"id\":1,\"name\":\"John
Doe\",\"email\":\"john@example.com\",\"roles\":[\"USER\",\"EDITOR\"]}";

            // Deserialize from JSON string
            User user = mapper.readValue(jsonString, User.class);
            System.out.println("User: " + user.getName() + ", Email: " +
user.getEmail());

            // Deserialize from file
            User userFromFile = mapper.readValue(new File("user.json"),
User.class);

            // Deserialize from byte array
            User userFromBytes = mapper.readValue(jsonBytes, User.class);

            // Deserialize from URL
            User userFromUrl = mapper.readValue(new
URL("http://example.com/api/user/1"), User.class);

        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

4. Working with JSON Nodes:

```
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;

public class JsonNodeExample {

    public static void main(String[] args) {
        try {
            // Create an ObjectMapper
            ObjectMapper mapper = new ObjectMapper();

            // Parse JSON into JsonNode
            String jsonString = "{\"id\":1,\"name\":\"John
Doe\",\"email\":\"john@example.com\",\"address\":{\"city\":\"New
York\",\"zipCode\":\"10001\"}}";
            JsonNode rootNode = mapper.readTree(jsonString);

            // Access JSON data
            long id = rootNode.get("id").asLong();
            String name = rootNode.get("name").asText();
            String email = rootNode.get("email").asText();

            System.out.println("ID: " + id);
            System.out.println("Name: " + name);
            System.out.println("Email: " + email);

            // Access nested data
            JsonNode addressNode = rootNode.get("address");
            if (addressNode != null) {
                String city = addressNode.get("city").asText();
                String zipCode = addressNode.get("zipCode").asText();

                System.out.println("City: " + city);
                System.out.println("Zip Code: " + zipCode);
            }

            // Check if a field exists
            if (rootNode.has("phone")) {
                String phone = rootNode.get("phone").asText();
                System.out.println("Phone: " + phone);
            } else {
                System.out.println("Phone field does not exist");
            }

            // Modify JSON data (needs to be ObjectNode)
```

```
            ObjectNode rootObjectNode = (ObjectNode) rootNode;
            rootObjectNode.put("phone", "555-1234");
            rootObjectNode.put("active", true);

            // Remove a field
            rootObjectNode.remove("email");

            // Convert back to JSON string
            String updatedJson = mapper.writeValueAsString(rootObjectNode);
            System.out.println(updatedJson);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5. Handling JSON Arrays:

```
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;

public class JsonArrayExample {

    public static void main(String[] args) {
        try {
            // Create an ObjectMapper
            ObjectMapper mapper = new ObjectMapper();

            // Parse JSON with array
            String jsonString = "{\"users\":[{\"id\":1,\"name\":\"John\"},
{\"id\":2,\"name\":\"Jane\"},{\"id\":3,\"name\":\"Bob\"}]}";
            JsonNode rootNode = mapper.readTree(jsonString);

            // Access array
            JsonNode usersNode = rootNode.get("users");

            if (usersNode.isArray()) {
                // Iterate through array
                for (JsonNode userNode : usersNode) {
                    long id = userNode.get("id").asLong();
                    String name = userNode.get("name").asText();

                    System.out.println("User ID: " + id + ", Name: " + name);
                }

                // Get array size
                int userCount = usersNode.size();
                System.out.println("User count: " + userCount);
```

```
                // Access by index
                JsonNode secondUser = usersNode.get(1);
                System.out.println("Second user: " +
secondUser.get("name").asText());
            }

            // Create a new array
            ArrayNode arrayNode = mapper.createArrayNode();
            ObjectNode user1 = mapper.createObjectNode();
            user1.put("id", 4);
            user1.put("name", "Alice");

            ObjectNode user2 = mapper.createObjectNode();
            user2.put("id", 5);
            user2.put("name", "Charlie");

            arrayNode.add(user1);
            arrayNode.add(user2);

            // Convert to JSON string
            String arrayJson = mapper.writeValueAsString(arrayNode);
            System.out.println(arrayJson);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

6. Custom Serialization and Deserialization:

```
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.databind.ser.std.StdSerializer;
import com.fasterxml.jackson.databind.JsonSerializer;
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.SerializerProvider;

public class CustomSerializationExample {

    // Custom date serializer
    static class DateSerializer extends StdSerializer<Date> {

        public DateSerializer() {
            this(null);
        }

        public DateSerializer(Class<Date> t) {
            super(t);
        }
```

```
        @Override
        public void serialize(Date date, JsonGenerator gen, SerializerProvider
provider)
                throws IOException {
            SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
            String formattedDate = formatter.format(date);
            gen.writeString(formattedDate);
        }
    }

    // Model with custom serialization
    static class Event {
        private long id;
        private String name;

        @JsonSerialize(using = DateSerializer.class)
        private Date eventDate;

        // Getters and setters
    }

    public static void main(String[] args) {
        try {
            // Create ObjectMapper
            ObjectMapper mapper = new ObjectMapper();

            // Create an event
            Event event = new Event();
            event.setId(1L);
            event.setName("Conference");
            event.setEventDate(new Date());

            // Serialize with custom serializer
            String json = mapper.writeValueAsString(event);
            System.out.println(json);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

7. Jackson in Servlets and JSPs:

```
@WebServlet("/json-servlet")
public class JsonServlet extends HttpServlet {

    private ObjectMapper mapper = new ObjectMapper();

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

```
        // Set content type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        // Create sample data
        User user = new User();
        user.setId(1L);
        user.setName("John Doe");
        user.setEmail("john@example.com");

        // Serialize to JSON and write to response
        mapper.writeValue(response.getWriter(), user);
    }

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

        // Parse JSON from request body
        User user = mapper.readValue(request.getInputStream(), User.class);

        // Process the user...

        // Send JSON response
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        Map<String, Object> responseData = new HashMap<>();
        responseData.put("status", "success");
        responseData.put("message", "User processed successfully");
        responseData.put("user", user);

        mapper.writeValue(response.getWriter(), responseData);
    }
}
```

8. Jackson Configuration:

```
public class JacksonConfig {

    public static ObjectMapper createObjectMapper() {
        ObjectMapper mapper = new ObjectMapper();

        // Configure mapper
        mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
        mapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false);
        mapper.setSerializationInclusion(JsonInclude.Include.NON_NULL);

        // Register modules
```

```
        mapper.registerModule(new JavaTimeModule());

        return mapper;
    }
}
```

Using JSON-P (JSON Processing API)

JSON-P is a standard Java API for JSON processing.

1. Adding JSON-P Dependencies:

```
<dependency>
    <groupId>javax.json</groupId>
    <artifactId>javax.json-api</artifactId>
    <version>1.1.4</version>
</dependency>
<dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>javax.json</artifactId>
    <version>1.1.4</version>
</dependency>
```

2. Reading JSON with JSON-P:

```
import javax.json.Json;
import javax.json.JsonArray;
import javax.json.JsonObject;
import javax.json.JsonReader;
import javax.json.JsonValue;
import java.io.StringReader;

public class JsonPReadExample {

    public static void main(String[] args) {
        // Sample JSON string
        String jsonString = "{\"id\":1,\"name\":\"John
Doe\",\"email\":\"john@example.com\",\"skills\":
[\"Java\",\"SQL\",\"JavaScript\"]}";

        try (JsonReader reader = Json.createReader(new StringReader(jsonString)))
{
            // Read JSON object
            JsonObject jsonObject = reader.readObject();

            // Access properties
            int id = jsonObject.getInt("id");
            String name = jsonObject.getString("name");
            String email = jsonObject.getString("email");
```

```
            System.out.println("ID: " + id);
            System.out.println("Name: " + name);
            System.out.println("Email: " + email);

            // Access array
            JsonArray skillsArray = jsonObject.getJsonArray("skills");
            System.out.println("Skills:");

            for (JsonValue skill : skillsArray) {
                System.out.println("- " + skill.toString().replace("\"", ""));
            }
        }
    }
}
```

3. Writing JSON with JSON-P:

```
import javax.json.Json;
import javax.json.JsonArray;
import javax.json.JsonArrayBuilder;
import javax.json.JsonObject;
import javax.json.JsonObjectBuilder;
import javax.json.JsonWriter;
import java.io.StringWriter;

public class JsonPWriteExample {

    public static void main(String[] args) {
        // Create JSON object using builder
        JsonObjectBuilder objectBuilder = Json.createObjectBuilder();

        objectBuilder.add("id", 1)
                     .add("name", "John Doe")
                     .add("email", "john@example.com")
                     .add("active", true);

        // Create nested object
        JsonObjectBuilder addressBuilder = Json.createObjectBuilder();
        addressBuilder.add("street", "123 Main St")
                      .add("city", "New York")
                      .add("zipCode", "10001");

        objectBuilder.add("address", addressBuilder);

        // Create array
        JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();
        arrayBuilder.add("Java")
                    .add("SQL")
                    .add("JavaScript");

        objectBuilder.add("skills", arrayBuilder);
```

```
        // Build the object
        JsonObject jsonObject = objectBuilder.build();

        // Write to string
        StringWriter stringWriter = new StringWriter();
        try (JsonWriter jsonWriter = Json.createWriter(stringWriter)) {
            jsonWriter.writeObject(jsonObject);
        }

        System.out.println(stringWriter.toString());
    }
}
```

4. JSON-P in Servlets:

```
@WebServlet("/jsonp-servlet")
public class JsonPServlet extends HttpServlet {

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

        // Set content type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        // Create JSON object
        JsonObjectBuilder objectBuilder = Json.createObjectBuilder();

        objectBuilder.add("id", 1)
                     .add("name", "John Doe")
                     .add("email", "john@example.com");

        JsonArrayBuilder rolesBuilder = Json.createArrayBuilder();
        rolesBuilder.add("USER")
                    .add("EDITOR");

        objectBuilder.add("roles", rolesBuilder);

        // Write JSON to response
        try (JsonWriter jsonWriter = Json.createWriter(response.getWriter())) {
            jsonWriter.writeObject(objectBuilder.build());
        }
    }

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

        // Read JSON from request
```

```
        try (JsonReader jsonReader = Json.createReader(request.getInputStream()))
{
            JsonObject jsonObject = jsonReader.readObject();

            // Process the JSON data
            String name = jsonObject.getString("name");
            String email = jsonObject.getString("email");

            // Create response
            JsonObjectBuilder responseBuilder = Json.createObjectBuilder();
            responseBuilder.add("status", "success")
                           .add("message", "Received data for: " + name);

            // Write response
            response.setContentType("application/json");
            try (JsonWriter jsonWriter = Json.createWriter(response.getWriter()))
{
                jsonWriter.writeObject(responseBuilder.build());
            }
        }
    }
}
```

Using JSON-B (JSON Binding)

JSON-B is a standard Java API for binding JSON documents to Java objects.

1. Adding JSON-B Dependencies:

```
<dependency>
    <groupId>javax.json.bind</groupId>
    <artifactId>javax.json.bind-api</artifactId>
    <version>1.0</version>
</dependency>
<dependency>
    <groupId>org.eclipse</groupId>
    <artifactId>yasson</artifactId>
    <version>1.0.2</version>
</dependency>
```

2. Serializing and Deserializing with JSON-B:

```
import javax.json.bind.Jsonb;
import javax.json.bind.JsonbBuilder;
import javax.json.bind.JsonbConfig;

public class JsonbExample {

    public static void main(String[] args) {
```

```
        try {
            // Create Jsonb instance
            Jsonb jsonb = JsonbBuilder.create();

            // Create a user
            User user = new User();
            user.setId(1L);
            user.setName("John Doe");
            user.setEmail("john@example.com");
            user.setRoles(new String[]{"USER", "EDITOR"});

            // Serialize to JSON
            String json = jsonb.toJson(user);
            System.out.println(json);

            // Deserialize from JSON
            User deserializedUser = jsonb.fromJson(json, User.class);
            System.out.println("Deserialized user: " +
deserializedUser.getName());

            // Configure Jsonb
            JsonbConfig config = new JsonbConfig()
                .withFormatting(true)
                .withNullValues(false);

            Jsonb configuredJsonb = JsonbBuilder.create(config);

            // Serialize with pretty printing
            String prettyJson = configuredJsonb.toJson(user);
            System.out.println(prettyJson);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. Using JSON-B Annotations:

```
import javax.json.bind.annotation.JsonbProperty;
import javax.json.bind.annotation.JsonbTransient;
import javax.json.bind.annotation.JsonbDateFormat;
import javax.json.bind.annotation.JsonbCreator;
import javax.json.bind.annotation.JsonbNillable;
import javax.json.bind.annotation.JsonbNumberFormat;

@JsonbNillable
public class Product {

    private Long id;

    @JsonbProperty("product_name")
```

```
    private String name;

    @JsonbNumberFormat("#,###.00")
    private BigDecimal price;

    @JsonbDateFormat("yyyy-MM-dd")
    private Date createdAt;

    @JsonbTransient
    private String internalCode;

    // Constructors, getters, and setters
}
```

4. JSON-B in Servlets:

```
@WebServlet("/jsonb-servlet")
public class JsonbServlet extends HttpServlet {

    private Jsonb jsonb;

    @Override
    public void init() throws ServletException {
        // Create Jsonb instance
        jsonb = JsonbBuilder.create();
    }

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

        // Set content type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        // Create sample data
        Product product = new Product();
        product.setId(1L);
        product.setName("Laptop");
        product.setPrice(new BigDecimal("999.99"));
        product.setCreatedAt(new Date());

        // Serialize to JSON and write to response
        jsonb.toJson(product, response.getWriter());
    }

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

        // Read JSON from request and convert to Product
```

```
        Product product = jsonb.fromJson(request.getInputStream(), Product.class);

        // Process the product...

        // Create response
        Map<String, Object> responseData = new HashMap<>();
        responseData.put("status", "success");
        responseData.put("message", "Product processed: " + product.getName());

        // Write response
        response.setContentType("application/json");
        jsonb.toJson(responseData, response.getWriter());
    }

    @Override
    public void destroy() {
        try {
            if (jsonb != null) {
                jsonb.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JSON Processing in JSP

1. Using Jackson in JSP:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ page import="com.fasterxml.jackson.databind.ObjectMapper" %>
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSON in JSP</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
    <h1>JSON in JSP Example</h1>

    <%
        // Create sample data
        Map<String, Object> user = new HashMap<>();
        user.put("id", 1);
        user.put("name", "John Doe");
        user.put("email", "john@example.com");
```

```
        List<String> roles = Arrays.asList("USER", "EDITOR");
        user.put("roles", roles);

        // Convert to JSON
        ObjectMapper mapper = new ObjectMapper();
        String userJson = mapper.writeValueAsString(user);
    %>

    <h2>User Data</h2>
    <pre id="userData"></pre>

    <h2>Dynamic Data Loading</h2>
    <button id="loadData">Load More Data</button>
    <div id="dynamicData"></div>

    <script>
        // Display user data
        const userData = document.getElementById('userData');
        const userObject = <%= userJson %>;
        userData.textContent = JSON.stringify(userObject, null, 2);

        // Dynamic data loading
        $(document).ready(function() {
            $('#loadData').click(function() {
                $.ajax({
                    url: 'json-servlet',
                    type: 'GET',
                    dataType: 'json',
                    success: function(data) {
                        $('#dynamicData').html('<pre>' + JSON.stringify(data,
null, 2) + '</pre>');
                    },
                    error: function(xhr, status, error) {
                        $('#dynamicData').html('<p>Error loading data: ' + error +
'</p>');
                    }
                });
            });
        });
    </script>
</body>
</html>
```

2. Using JSON with JSTL:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<!DOCTYPE html>
<html>
<head>
```

```
    <meta charset="UTF-8">
    <title>JSTL and JSON</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
    <h1>JSTL and JSON Example</h1>

    <%-- Assume users attribute contains a JSON string of users --%>
    <c:set var="usersJson" value='${users}' />

    <h2>Users Table</h2>
    <table border="1">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Email</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody id="usersTableBody">
            <%-- Will be populated by JavaScript --%>
        </tbody>
    </table>

    <script>
        // Parse JSON data
        const usersData = ${usersJson};

        // Populate table
        const tableBody = document.getElementById('usersTableBody');

        usersData.forEach(user => {
            const row = document.createElement('tr');

            const idCell = document.createElement('td');
            idCell.textContent = user.id;
            row.appendChild(idCell);

            const nameCell = document.createElement('td');
            nameCell.textContent = user.name;
            row.appendChild(nameCell);

            const emailCell = document.createElement('td');
            emailCell.textContent = user.email;
            row.appendChild(emailCell);

            const actionsCell = document.createElement('td');
            const editLink = document.createElement('a');
            editLink.href = 'edit-user?id=' + user.id;
            editLink.textContent = 'Edit';
            actionsCell.appendChild(editLink);

            actionsCell.appendChild(document.createTextNode(' | '));
```

```
            const deleteLink = document.createElement('a');
            deleteLink.href = '#';
            deleteLink.textContent = 'Delete';
            deleteLink.onclick = function() {
                if (confirm('Delete this user?')) {
                    deleteUser(user.id);
                }
                return false;
            };
            actionsCell.appendChild(deleteLink);

            row.appendChild(actionsCell);

            tableBody.appendChild(row);
        });

        function deleteUser(userId) {
            $.ajax({
                url: 'delete-user?id=' + userId,
                type: 'POST',
                success: function(response) {
                    if (response.success) {
                        alert('User deleted successfully');
                        location.reload();
                    } else {
                        alert('Error: ' + response.message);
                    }
                },
                error: function() {
                    alert('An error occurred');
                }
            });
        }
    </script>
</body>
</html>
```

JSON Processing Best Practices

1. **Choose the Right Library**:
    - Jackson for full-featured JSON processing
    - JSON-P for low-level JSON manipulation
    - JSON-B for simple Java object binding
2. **Performance Considerations**:
    - Reuse ObjectMapper instances
    - Use streaming API for large documents
    - Configure serialization/deserialization for optimal performance

3. **Error Handling**:
    - Handle parsing exceptions gracefully
    - Provide meaningful error messages
    - Validate JSON structure before processing
4. **Security Best Practices**:
    - Validate all JSON input
    - Use explicit typing for deserialization when possible
    - Be cautious with dynamic property deserializatio
5. **Date and Time Handling**:
    - Use consistent date/time formats
    - Consider time zones
    - Use Java 8 Date/Time API with appropriate serializers
6. **Custom Serialization**:
    - Implement custom serializers for complex types
    - Use annotations to control serialization behavior
    - Exclude sensitive or unnecessary fields
7. **Testing**:
    - Test serialization/deserialization with edge cases
    - Verify round-trip conversions (Java -> JSON -> Java)
    - Test with malformed input

5.4 Integrating with Frontend Frameworks

Modern web applications often use JavaScript frontend frameworks that communicate with Java backend services via APIs. Here's how to integrate Servlets and JSP with popular frontend frameworks.

Serving Static Frontend Files from a Servlet Application

1. Configure Your Web Application to Serve Static Content:

```
<!-- web.xml -->
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/static/*</url-pattern>
</servlet-mapping>
```

2. Organize Your Static Files:

```
webapp/
├── WEB-INF/
```

```
│   ├── web.xml
│   └── ...
├── static/
│   ├── css/
│   │   └── styles.css
│   ├── js/
│   │   └── app.js
│   └── index.html
└── ...
```

3. Create a Simple Frontend Page:

```html
<!-- webapp/static/index.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Java EE Frontend Integration</title>
    <link rel="stylesheet" href="css/styles.css" />
  </head>
  <body>
    <div id="app">
      <h1>Java EE Frontend Integration</h1>
      <div id="content">
        <p>Loading data...</p>
      </div>
      <button id="loadButton">Load Data</button>
    </div>

    <script src="js/app.js"></script>
  </body>
</html>
```

```css
/* webapp/static/css/styles.css */
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  background-color: #f5f5f5;
}

#app {
  max-width: 800px;
  margin: 0 auto;
  background-color: #fff;
  padding: 20px;
  border-radius: 5px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}
```

```
button {
  padding: 8px 16px;
  background-color: #4caf50;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
  background-color: #45a049;
}
```

```
// webapp/static/js/app.js
document.addEventListener('DOMContentLoaded', function () {
  const contentDiv = document.getElementById('content');
  const loadButton = document.getElementById('loadButton');

  loadButton.addEventListener('click', function () {
    fetch('/api/users')
      .then((response) => {
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        return response.json();
      })
      .then((data) => {
        displayUsers(data);
      })
      .catch((error) => {
        contentDiv.innerHTML = `<p>Error: ${error.message}</p>`;
      });
  });

  function displayUsers(users) {
    if (!users || users.length === 0) {
      contentDiv.innerHTML = '<p>No users found</p>';
      return;
    }

    let html = '<h2>Users</h2><ul>';
    users.forEach((user) => {
      html += `<li>
                <strong>${user.name}</strong> (${user.email})
                <button class="edit-btn" data-id="${user.id}">Edit</button>
                <button class="delete-btn" data-id="${user.id}">Delete</button>
              </li>`;
    });
    html += '</ul>';
```

```
      contentDiv.innerHTML = html;

      // Add event listeners to dynamically created buttons
      document.querySelectorAll('.edit-btn').forEach((button) => {
        button.addEventListener('click', function () {
          const userId = this.getAttribute('data-id');
          editUser(userId);
        });
      });

      document.querySelectorAll('.delete-btn').forEach((button) => {
        button.addEventListener('click', function () {
          const userId = this.getAttribute('data-id');
          deleteUser(userId);
        });
      });
    }

    function editUser(userId) {
      // Fetch user details and show edit form
      fetch(`/api/users/${userId}`)
        .then((response) => response.json())
        .then((user) => {
          const formHtml = `
                      <h2>Edit User</h2>
                      <form id="editForm">
                          <input type="hidden" id="userId" value="${user.id}">
                          <div>
                              <label for="name">Name:</label>
                              <input type="text" id="name" value="${user.name}"
required>
                          </div>
                          <div>
                              <label for="email">Email:</label>
                              <input type="email" id="email" value="${user.email}"
required>
                          </div>
                          <div>
                              <button type="submit">Save</button>
                              <button type="button" id="cancelBtn">Cancel</button>
                          </div>
                      </form>
                  `;

          contentDiv.innerHTML = formHtml;

          // Add form submission handler
          document
            .getElementById('editForm')
            .addEventListener('submit', function (e) {
              e.preventDefault();
              updateUser();
            });
```

```
        // Add cancel button handler
        document
          .getElementById('cancelBtn')
          .addEventListener('click', function () {
            loadButton.click(); // Reload user list
          });
      })
      .catch((error) => {
        contentDiv.innerHTML = `<p>Error loading user: ${error.message}</p>`;
      });
  }

  function updateUser() {
    const userId = document.getElementById('userId').value;
    const name = document.getElementById('name').value;
    const email = document.getElementById('email').value;

    fetch(`/api/users/${userId}`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ id: userId, name, email }),
    })
      .then((response) => response.json())
      .then((data) => {
        alert('User updated successfully');
        loadButton.click(); // Reload user list
      })
      .catch((error) => {
        alert('Error updating user: ' + error.message);
      });
  }

  function deleteUser(userId) {
    if (confirm('Are you sure you want to delete this user?')) {
      fetch(`/api/users/${userId}`, {
        method: 'DELETE',
      })
        .then((response) => {
          if (response.ok || response.status === 204) {
            alert('User deleted successfully');
            loadButton.click(); // Reload user list
          } else {
            return response.json().then((data) => {
              throw new Error(data.error || 'Failed to delete user');
            });
          }
        })
        .catch((error) => {
          alert('Error: ' + error.message);
        });
    }
```

```
  }
});
```

Integrating with React

1. Setup a React Frontend:

Create a React application with create-react-app:

```
npx create-react-app frontend
cd frontend
```

2. Configure Proxy for Development:

In package.json, add a proxy to your Java backend:

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    // ...
  },
  "proxy": "http://localhost:8080",
  "scripts": {
    // ...
  }
}
```

3. Create React Components:

```
// src/App.js
import React, { useState, useEffect } from 'react';
import UserList from './components/UserList';
import UserForm from './components/UserForm';
import './App.css';

function App() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const [selectedUser, setSelectedUser] = useState(null);

  const fetchUsers = async () => {
    setLoading(true);
    try {
      const response = await fetch('/api/users');
```

```
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      const data = await response.json();
      setUsers(data);
      setError(null);
    } catch (e) {
      setError('Failed to fetch users: ' + e.message);
      setUsers([]);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchUsers();
  }, []);

  const handleEditUser = (user) => {
    setSelectedUser(user);
  };

  const handleDeleteUser = async (userId) => {
    if (window.confirm('Are you sure you want to delete this user?')) {
      try {
        const response = await fetch(`/api/users/${userId}`, {
          method: 'DELETE',
        });
        if (!response.ok && response.status !== 204) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }
        setUsers(users.filter((user) => user.id !== userId));
        alert('User deleted successfully');
      } catch (e) {
        alert('Failed to delete user: ' + e.message);
      }
    }
  };

  const handleSaveUser = async (userData) => {
    try {
      const isNewUser = !userData.id;
      const url = isNewUser ? '/api/users' : `/api/users/${userData.id}`;
      const method = isNewUser ? 'POST' : 'PUT';

      const response = await fetch(url, {
        method,
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(userData),
      });

      if (!response.ok) {
```

```
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      const savedUser = await response.json();

      if (isNewUser) {
        setUsers([...users, savedUser]);
      } else {
        setUsers(
          users.map((user) => (user.id === savedUser.id ? savedUser : user))
        );
      }

      setSelectedUser(null);
      alert(`User ${isNewUser ? 'created' : 'updated'} successfully`);
    } catch (e) {
      alert('Failed to save user: ' + e.message);
    }
  };

  const handleCancelEdit = () => {
    setSelectedUser(null);
  };

  return (
    <div className="App">
      <header className="App-header">
        <h1>User Management</h1>
      </header>

      <main>
        {loading && <p>Loading...</p>}
        {error && <p className="error">{error}</p>}

        {!loading && !error && (
          <>
            <UserList
              users={users}
              onEditUser={handleEditUser}
              onDeleteUser={handleDeleteUser}
            />

            <UserForm
              user={selectedUser}
              onSave={handleSaveUser}
              onCancel={handleCancelEdit}
            />
          </>
        )}
      </main>
    </div>
  );
}
```

```
export default App;
```

```
// src/components/UserList.js
import React from 'react';

function UserList({ users, onEditUser, onDeleteUser }) {
  if (!users || users.length === 0) {
    return <p>No users found.</p>;
  }

  return (
    <div className="user-list">
      <h2>Users</h2>
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {users.map((user) => (
            <tr key={user.id}>
              <td>{user.id}</td>
              <td>{user.name}</td>
              <td>{user.email}</td>
              <td>
                <button onClick={() => onEditUser(user)}>Edit</button>
                <button onClick={() => onDeleteUser(user.id)}>Delete</button>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}

export default UserList;
```

```
// src/components/UserForm.js
import React, { useState, useEffect } from 'react';

function UserForm({ user, onSave, onCancel }) {
  const [formData, setFormData] = useState({
    id: '',
```

```
    name: '',
    email: '',
  });

  useEffect(() => {
    if (user) {
      setFormData(user);
    } else {
      setFormData({ id: '', name: '', email: '' });
    }
  }, [user]);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prevData) => ({
      ...prevData,
      [name]: value,
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    onSave(formData);
  };

  return (
    <div className="user-form">
      <h2>{formData.id ? 'Edit User' : 'Add User'}</h2>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label htmlFor="name">Name:</label>
          <input
            type="text"
            id="name"
            name="name"
            value={formData.name}
            onChange={handleChange}
            required
          />
        </div>

        <div className="form-group">
          <label htmlFor="email">Email:</label>
          <input
            type="email"
            id="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
            required
          />
        </div>

        <div className="form-actions">
```

```
            <button type="submit">Save</button>
            {formData.id && (
              <button type="button" onClick={onCancel}>
                Cancel
              </button>
            )}
          </div>
        </form>
      </div>
    );
}

export default UserForm;
```

4. Build and Deploy with Servlets:

Create a build script to integrate React with your Java web application:

```
# Build the React application
cd frontend
npm run build

# Copy the built files to your Java web application
cp -R build/* ../src/main/webapp/static/
```

5. Configure Your Servlets to Handle API Requests and Static Content:

```
@WebServlet("/api/*")
public class ApiServlet extends HttpServlet {
    // API implementation...
}

@WebServlet(urlPatterns = {"/", "/users", "/users/*"})
public class FrontendServlet extends HttpServlet {

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

        // Forward all requests to the React single page application
        request.getRequestDispatcher("/static/index.html").forward(request,
response);
    }
}
```

Integrating with Angular

1. Setup an Angular Frontend:

Create an Angular application:

```
ng new frontend
cd frontend
```

2. Configure Proxy for Development:

Create a proxy.conf.json file:

```
{
  "/api": {
    "target": "http://localhost:8080",
    "secure": false
  }
}
```

Update angular.json to use the proxy:

```
"architect": {
  "serve": {
    "builder": "@angular-devkit/build-angular:dev-server",
    "options": {
      "browserTarget": "frontend:build",
      "proxyConfig": "proxy.conf.json"
    },
    // ...
  }
}
```

3. Create Angular Components and Services:

```
// src/app/models/user.model.ts
export interface User {
  id?: number;
  name: string;
  email: string;
}
```

```
// src/app/services/user.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from '../models/user.model';
```

```
@Injectable({
  providedIn: 'root',
})
export class UserService {
  private apiUrl = '/api/users';

  constructor(private http: HttpClient) {}

  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.apiUrl);
  }

  getUserById(id: number): Observable<User> {
    return this.http.get<User>(`${this.apiUrl}/${id}`);
  }

  createUser(user: User): Observable<User> {
    return this.http.post<User>(this.apiUrl, user);
  }

  updateUser(user: User): Observable<User> {
    return this.http.put<User>(`${this.apiUrl}/${user.id}`, user);
  }

  deleteUser(id: number): Observable<any> {
    return this.http.delete(`${this.apiUrl}/${id}`);
  }
}
```

```
// src/app/components/user-list/user-list.component.ts
import { Component, OnInit } from '@angular/core';
import { User } from '../../models/user.model';
import { UserService } from '../../services/user.service';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css'],
})
export class UserListComponent implements OnInit {
  users: User[] = [];
  loading = false;
  error: string | null = null;

  constructor(private userService: UserService) {}

  ngOnInit(): void {
    this.loadUsers();
  }

  loadUsers(): void {
```

```
    this.loading = true;
    this.error = null;

    this.userService.getUsers().subscribe(
      (data) => {
        this.users = data;
        this.loading = false;
      },
      (error) => {
        this.error = 'Failed to load users: ' + error.message;
        this.loading = false;
      }
    );
  }

  deleteUser(id: number): void {
    if (confirm('Are you sure you want to delete this user?')) {
      this.userService.deleteUser(id).subscribe(
        () => {
          this.users = this.users.filter((user) => user.id !== id);
          alert('User deleted successfully');
        },
        (error) => {
          alert('Failed to delete user: ' + error.message);
        }
      );
    }
  }
}
```

```
<!-- src/app/components/user-list/user-list.component.html -->
<div class="user-list">
  <h2>Users</h2>

  <div *ngIf="loading">Loading...</div>
  <div *ngIf="error" class="error">{{ error }}</div>

  <table *ngIf="!loading && !error && users.length > 0">
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Email</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let user of users">
        <td>{{ user.id }}</td>
        <td>{{ user.name }}</td>
        <td>{{ user.email }}</td>
```

```
        <td>
          <button [routerLink]="['/users/edit', user.id]">Edit</button>
          <button (click)="deleteUser(user.id)">Delete</button>
        </td>
      </tr>
    </tbody>
  </table>

  <div *ngIf="!loading && !error && users.length === 0">No users found.</div>

  <button [routerLink]="['/users/add']">Add New User</button>
</div>
```

4. Build and Deploy with Servlets:

```
# Build the Angular application
cd frontend
ng build --prod

# Copy the built files to your Java web application
cp -R dist/frontend/* ../src/main/webapp/static/
```

Integrating with Vue.js

1. Setup a Vue.js Frontend:

Create a Vue.js application:

```
vue create frontend
cd frontend
```

2. Configure Proxy for Development:

Create a vue.config.js file:

```
module.exports = {
  devServer: {
    proxy: {
      '/api': {
        target: 'http://localhost:8080',
        ws: true,
        changeOrigin: true,
      },
    },
  },
};
```

3. Create Vue Components:

```
// src/services/UserService.js
import axios from 'axios';

const API_URL = '/api/users';

class UserService {
  getUsers() {
    return axios.get(API_URL);
  }

  getUserById(id) {
    return axios.get(`${API_URL}/${id}`);
  }

  createUser(user) {
    return axios.post(API_URL, user);
  }

  updateUser(id, user) {
    return axios.put(`${API_URL}/${id}`, user);
  }

  deleteUser(id) {
    return axios.delete(`${API_URL}/${id}`);
  }
}

export default new UserService();
```

```
<!-- src/components/UserList.vue -->
<template>
  <div class="user-list">
    <h2>Users</h2>

    <div v-if="loading" class="loading">Loading...</div>
    <div v-if="error" class="error">{{ error }}</div>

    <table v-if="!loading && !error && users.length > 0">
      <thead>
        <tr>
          <th>ID</th>
          <th>Name</th>
          <th>Email</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="user in users" :key="user.id">
```

```
            <td>{{ user.id }}</td>
            <td>{{ user.name }}</td>
            <td>{{ user.email }}</td>
            <td>
              <button @click="editUser(user)">Edit</button>
              <button @click="deleteUser(user.id)">Delete</button>
            </td>
          </tr>
        </tbody>
      </table>

      <div v-if="!loading && !error && users.length === 0" class="no-data">
        No users found.
      </div>

      <button @click="showAddForm = true">Add New User</button>

      <!-- User Form Modal -->
      <div v-if="showAddForm || selectedUser" class="modal">
        <div class="modal-content">
          <span class="close" @click="closeForm">&times;</span>
          <h3>{{ selectedUser ? 'Edit User' : 'Add User' }}</h3>

          <form @submit.prevent="saveUser">
            <div class="form-group">
              <label for="name">Name:</label>
              <input type="text" id="name" v-model="formData.name" required />
            </div>

            <div class="form-group">
              <label for="email">Email:</label>
              <input type="email" id="email" v-model="formData.email" required />
            </div>

            <div class="form-actions">
              <button type="submit">Save</button>
              <button type="button" @click="closeForm">Cancel</button>
            </div>
          </form>
        </div>
      </div>
    </div>
  </template>

  <script>
  import UserService from '../services/UserService';

  export default {
    data() {
      return {
        users: [],
        loading: false,
        error: null,
        showAddForm: false,
```

```
      selectedUser: null,
      formData: {
        name: '',
        email: '',
      },
    };
  },
  created() {
    this.loadUsers();
  },
  methods: {
    loadUsers() {
      this.loading = true;
      this.error = null;

      UserService.getUsers()
        .then((response) => {
          this.users = response.data;
          this.loading = false;
        })
        .catch((error) => {
          this.error = 'Failed to load users: ' + error.message;
          this.loading = false;
        });
    },
    editUser(user) {
      this.selectedUser = user;
      this.formData = { ...user };
    },
    deleteUser(id) {
      if (confirm('Are you sure you want to delete this user?')) {
        UserService.deleteUser(id)
          .then(() => {
            this.users = this.users.filter((user) => user.id !== id);
            alert('User deleted successfully');
          })
          .catch((error) => {
            alert('Failed to delete user: ' + error.message);
          });
      }
    },
    saveUser() {
      const isNewUser = !this.selectedUser;
      const promise = isNewUser
        ? UserService.createUser(this.formData)
        : UserService.updateUser(this.selectedUser.id, this.formData);

      promise
        .then((response) => {
          if (isNewUser) {
            this.users.push(response.data);
          } else {
            const index = this.users.findIndex(
              (u) => u.id === this.selectedUser.id
```

```
            );
            if (index !== -1) {
              this.users.splice(index, 1, response.data);
            }
          }
          this.closeForm();
          alert(`User ${isNewUser ? 'created' : 'updated'} successfully`);
        })
        .catch((error) => {
          alert(
            `Failed to ${isNewUser ? 'create' : 'update'} user: ` +
              error.message
          );
        });
    },
    closeForm() {
      this.showAddForm = false;
      this.selectedUser = null;
      this.formData = { name: '', email: '' };
    },
  },
};
</script>

<style scoped>
.user-list {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 20px;
}

th,
td {
  border: 1px solid #ddd;
  padding: 8px;
  text-align: left;
}

th {
  background-color: #f2f2f2;
}

.modal {
  position: fixed;
  z-index: 1;
  left: 0;
  top: 0;
  width: 100%;
```

```
    height: 100%;
    background-color: rgba(0, 0, 0, 0.4);
    display: flex;
    align-items: center;
    justify-content: center;
  }

  .modal-content {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    width: 60%;
    max-width: 500px;
  }

  .close {
    float: right;
    font-size: 28px;
    font-weight: bold;
    cursor: pointer;
  }

  .form-group {
    margin-bottom: 15px;
  }

  .form-group label {
    display: block;
    margin-bottom: 5px;
  }

  .form-group input {
    width: 100%;
    padding: 8px;
    border: 1px solid #ddd;
    border-radius: 4px;
  }

  .form-actions {
    margin-top: 20px;
  }

  button {
    padding: 8px 16px;
    margin-right: 10px;
    background-color: #4caf50;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
  }

  button:hover {
    background-color: #45a049;
```

```
}

.loading,
.error,
.no-data {
  padding: 20px;
  text-align: center;
}

.error {
  color: red;
}
</style>
```

4. Build and Deploy with Servlets:

```
# Build the Vue.js application
cd frontend
npm run build

# Copy the built files to your Java web application
cp -R dist/* ../src/main/webapp/static/
```

Building a Single Page Application (SPA) with Servlet Backend

Here's how to set up a servlet-based API that works well with single-page applications:

1. Create a Servlet Filter for API Requests:

```
@WebFilter("/api/*")
public class ApiFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Set common headers for API responses
        httpResponse.setHeader("Cache-Control", "no-cache, no-store, must-
revalidate");
        httpResponse.setHeader("Pragma", "no-cache");
        httpResponse.setHeader("Expires", "0");

        // Add CORS headers if needed
        httpResponse.setHeader("Access-Control-Allow-Origin", "*");
        httpResponse.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT,
```

```
DELETE, OPTIONS");
        httpResponse.setHeader("Access-Control-Allow-Headers", "Content-Type,
Authorization");

        // Handle preflight OPTIONS requests
        if (httpRequest.getMethod().equals("OPTIONS")) {
            httpResponse.setStatus(HttpServletResponse.SC_OK);
            return;
        }

        // Continue with request
        chain.doFilter(request, response);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

2. Create a URL Rewrite Filter for SPAs:

```
@WebFilter(urlPatterns = {"/*"})
public class SpaRewriteFilter implements Filter {

    private Set<String> excludedPaths = new HashSet<>(Arrays.asList(
        "/api/",
        "/static/",
        "/assets/",
        "/css/",
        "/js/",
        "/img/"
    ));

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        String path =
httpRequest.getRequestURI().substring(httpRequest.getContextPath().length());

        // Check if the request is for an API endpoint or static resource
        boolean shouldRedirect = true;
        for (String excludedPath : excludedPaths) {
            if (path.startsWith(excludedPath)) {
```

```
                shouldRedirect = false;
                break;
            }
        }

        // Check for file extensions that indicate static resources
        if (path.contains(".")) {
            String extension = path.substring(path.lastIndexOf("."));
            if (extension.matches("\\.
(html|js|css|png|jpg|jpeg|gif|svg|ico|woff|woff2|ttf|eot)$")) {
                shouldRedirect = false;
            }
        }

        if (shouldRedirect) {
            // Forward to the index.html for SPA routing
            request.getRequestDispatcher("/static/index.html").forward(request,
response);
        } else {
            // Continue with the normal request processing
            chain.doFilter(request, response);
        }
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

3. Set Up a RestServlet Base Class:

```
public abstract class RestServlet extends HttpServlet {

    private ObjectMapper objectMapper = new ObjectMapper();

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

        // Set default content type
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        try {
            // Call parent service method
```

```
            super.service(request, response);
        } catch (Exception e) {
            // Handle exceptions
            handleException(e, response);
        }
    }

    protected <T> T readRequest(HttpServletRequest request, Class<T> clazz) throws
IOException {
        return objectMapper.readValue(request.getInputStream(), clazz);
    }

    protected void writeResponse(HttpServletResponse response, Object data) throws
IOException {
        objectMapper.writeValue(response.getWriter(), data);
    }

    protected void writeResponse(HttpServletResponse response, Object data, int
statusCode) throws IOException {
        response.setStatus(statusCode);
        objectMapper.writeValue(response.getWriter(), data);
    }

    protected void handleException(Exception e, HttpServletResponse response)
throws IOException {
        if (e instanceof IllegalArgumentException) {
            // Handle validation errors
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            Map<String, String> error = new HashMap<>();
            error.put("error", e.getMessage());
            objectMapper.writeValue(response.getWriter(), error);
        } else {
            // Handle other errors
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            Map<String, String> error = new HashMap<>();
            error.put("error", "An unexpected error occurred");
            objectMapper.writeValue(response.getWriter(), error);

            // Log the exception
            getServletContext().log("Error in RestServlet", e);
        }
    }

    protected Long getIdFromPath(HttpServletRequest request) {
        String pathInfo = request.getPathInfo();
        if (pathInfo == null || pathInfo.equals("/")) {
            return null;
        }

        try {
            return Long.parseLong(pathInfo.substring(1));
        } catch (NumberFormatException e) {
            return null;
        }
```

```
    }
}
```

4. Create Specific API Servlets:

```
@WebServlet("/api/users/*")
public class UserApiServlet extends RestServlet {

    private UserService userService = new UserService();

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

        Long id = getIdFromPath(request);

        if (id == null) {
            // Get all users
            List<User> users = userService.getAllUsers();
            writeResponse(response, users);
        } else {
            // Get user by ID
            User user = userService.getUserById(id);

            if (user == null) {
                response.setStatus(HttpServletResponse.SC_NOT_FOUND);
                Map<String, String> error = new HashMap<>();
                error.put("error", "User not found");
                writeResponse(response, error);
            } else {
                writeResponse(response, user);
            }
        }
    }

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

        User user = readRequest(request, User.class);

        // Validate user
        if (user.getName() == null || user.getName().trim().isEmpty()) {
            throw new IllegalArgumentException("Name is required");
        }

        if (user.getEmail() == null || user.getEmail().trim().isEmpty()) {
            throw new IllegalArgumentException("Email is required");
        }

        // Save user
```

```
        userService.addUser(user);

        // Return created user
        writeResponse(response, user, HttpServletResponse.SC_CREATED);
    }

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

        Long id = getIdFromPath(request);

        if (id == null) {
            throw new IllegalArgumentException("User ID is required");
        }

        User existingUser = userService.getUserById(id);
        if (existingUser == null) {
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            Map<String, String> error = new HashMap<>();
            error.put("error", "User not found");
            writeResponse(response, error);
            return;
        }

        User updatedUser = readRequest(request, User.class);
        updatedUser.setId(id);

        // Validate user
        if (updatedUser.getName() == null ||
updatedUser.getName().trim().isEmpty()) {
            throw new IllegalArgumentException("Name is required");
        }

        if (updatedUser.getEmail() == null ||
updatedUser.getEmail().trim().isEmpty()) {
            throw new IllegalArgumentException("Email is required");
        }

        // Update user
        userService.updateUser(updatedUser);

        // Return updated user
        writeResponse(response, updatedUser);
    }

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

        Long id = getIdFromPath(request);

        if (id == null) {
```

```
            throw new IllegalArgumentException("User ID is required");
        }

        User existingUser = userService.getUserById(id);
        if (existingUser == null) {
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            Map<String, String> error = new HashMap<>();
            error.put("error", "User not found");
            writeResponse(response, error);
            return;
        }

        // Delete user
        userService.deleteUser(id);

        // Return no content
        response.setStatus(HttpServletResponse.SC_NO_CONTENT);
    }
}
```

Frontend Integration Best Practices

1. **Choose the Right Architecture**:
    - Use REST APIs for frontend-backend communication
    - Implement clear API contracts
    - Follow SPA (Single Page Application) patterns for modern UIs
2. **Security Considerations**:
    - Implement proper CORS configuration
    - Use CSRF protection
    - Secure API endpoints with authentication and authorization
    - Use HTTPS for all communications
3. **Performance Optimization**:
    - Minify and bundle frontend assets
    - Use proper caching headers
    - Implement lazy loading for JavaScript modules
    - Optimize API responses for size and speed
4. **Build and Deployment**:
    - Automate the frontend build process
    - Integrate frontend and backend builds
    - Implement CI/CD pipelines
    - Consider containerization for consistency
5. **Development Workflow**:

    - Use proxy configuration for local development
    - Set up hot-reloading for faster development
    - Implement proper error handling and logging
    - Use feature flags for gradual rollouts

6. **Testing Strategy**:

    - Implement frontend unit tests
    - Add integration tests for API endpoints
    - Test frontend-backend interactions
    - Use end-to-end testing for critical flows

7. **Error Handling**:

    - Implement consistent error responses
    - Provide meaningful error messages
    - Handle network errors gracefully
    - Add fallback mechanisms for critical features

6. Comprehensive Project: Task Management System

Now let's apply all the concepts we've learned to build a complete web application: a Task Management System. This project will demonstrate how to integrate Servlets, JSP, JDBC, and various patterns and technologies covered in this guide.

6.1 Project Overview

Task Management System Features:

- User Registration and Authentication
- Task Creation, Reading, Updating, and Deletion (CRUD)
- Task Assignment to Users
- Task Categories and Status Management
- Task Filtering and Searching
- RESTful API for Mobile Integration
- Admin Dashboard with Statistics

Technologies Used:

- Servlets and JSP
- JDBC with Connection Pooling
- MVC Architecture
- DAO Pattern
- Filter-based Authentication
- Custom Tag Libraries
- JSON Processing
- Front-end Integration (jQuery and Bootstrap)

6.2 Project Setup

1. Database Schema

```
-- Users Table
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    role ENUM('ADMIN', 'USER') NOT NULL DEFAULT 'USER',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Categories Table
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    color VARCHAR(20) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Tasks Table
CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    due_date DATE,
    priority ENUM('LOW', 'MEDIUM', 'HIGH') NOT NULL DEFAULT 'MEDIUM',
    status ENUM('TODO', 'IN_PROGRESS', 'DONE') NOT NULL DEFAULT 'TODO',
    category_id INT,
    creator_id INT NOT NULL,
    assignee_id INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(id) ON DELETE SET NULL,
    FOREIGN KEY (creator_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (assignee_id) REFERENCES users(id) ON DELETE SET NULL
);

-- Task Comments Table
CREATE TABLE task_comments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    task_id INT NOT NULL,
    user_id INT NOT NULL,
    comment TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (task_id) REFERENCES tasks(id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

-- Task Attachments Table
```

```sql
CREATE TABLE task_attachments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    task_id INT NOT NULL,
    file_name VARCHAR(255) NOT NULL,
    file_type VARCHAR(100) NOT NULL,
    file_size INT NOT NULL,
    uploaded_by INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (task_id) REFERENCES tasks(id) ON DELETE CASCADE,
    FOREIGN KEY (uploaded_by) REFERENCES users(id) ON DELETE CASCADE
);

-- Initial Data
INSERT INTO categories (name, color) VALUES
('Work', '#ff5722'),
('Personal', '#2196f3'),
('Study', '#4caf50'),
('Urgent', '#f44336'),
('Long-term', '#9c27b0');
```

2. Project Structure

```
task-manager/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/
│       │       └── taskmanager/
│       │           ├── config/
│       │           │   ├── DBConnectionPool.java
│       │           │   └── AppConfig.java
│       │           ├── dao/
│       │           │   ├── UserDAO.java
│       │           │   ├── TaskDAO.java
│       │           │   ├── CategoryDAO.java
│       │           │   └── impl/
│       │           │       ├── JdbcUserDAO.java
│       │           │       ├── JdbcTaskDAO.java
│       │           │       └── JdbcCategoryDAO.java
│       │           ├── model/
│       │           │   ├── User.java
│       │           │   ├── Task.java
│       │           │   ├── Category.java
│       │           │   ├── Comment.java
│       │           │   └── Attachment.java
│       │           ├── service/
│       │           │   ├── UserService.java
│       │           │   ├── TaskService.java
│       │           │   ├── CategoryService.java
│       │           │   └── AuthService.java
│       │           ├── util/
```

```
    │       │           │   ├── PasswordUtil.java
    │       │           │   └── ValidationUtil.java
    │       │           ├── web/
    │       │           │   ├── filter/
    │       │           │   │   ├── AuthenticationFilter.java
    │       │           │   │   └── LoggingFilter.java
    │       │           │   ├── listener/
    │       │           │   │   └── AppInitListener.java
    │       │           │   ├── servlet/
    │       │           │   │   ├── auth/
    │       │           │   │   │   ├── LoginServlet.java
    │       │           │   │   │   ├── LogoutServlet.java
    │       │           │   │   │   └── RegisterServlet.java
    │       │           │   │   ├── task/
    │       │           │   │   │   ├── TaskListServlet.java
    │       │           │   │   │   ├── TaskFormServlet.java
    │       │           │   │   │   ├── TaskDetailServlet.java
    │       │           │   │   │   └── TaskDeleteServlet.java
    │       │           │   │   ├── category/
    │       │           │   │   │   ├── CategoryListServlet.java
    │       │           │   │   │   └── CategoryFormServlet.java
    │       │           │   │   ├── user/
    │       │           │   │   │   ├── UserListServlet.java
    │       │           │   │   │   ├── UserProfileServlet.java
    │       │           │   │   │   └── UserFormServlet.java
    │       │           │   │   └── api/
    │       │           │   │       ├── TaskApiServlet.java
    │       │           │   │       ├── CategoryApiServlet.java
    │       │           │   │       └── UserApiServlet.java
    │       │           │   └── tag/
    │       │           │       ├── PaginationTag.java
    │       │           │       └── FormatDateTag.java
    │       │           └── exception/
    │       │               ├── ServiceException.java
    │       │               ├── DAOException.java
    │       │               ├── AuthenticationException.java
    │       │               └── ValidationException.java
    │       ├── resources/
    │       │   ├── log4j.properties
    │       │   └── db.properties
    │       └── webapp/
    │           ├── WEB-INF/
    │           │   ├── web.xml
    │           │   ├── tags/
    │           │   │   ├── pagination.tag
    │           │   │   └── formatDate.tag
    │           │   ├── taglib/
    │           │   │   └── taskmanager.tld
    │           │   └── views/
    │           │       ├── common/
    │           │       │   ├── header.jsp
    │           │       │   ├── footer.jsp
    │           │       │   ├── navigation.jsp
    │           │       │   └── error.jsp
```

```
│           │       ├── auth/
│           │       │   ├── login.jsp
│           │       │   └── register.jsp
│           │       ├── task/
│           │       │   ├── list.jsp
│           │       │   ├── form.jsp
│           │       │   └── detail.jsp
│           │       ├── category/
│           │       │   ├── list.jsp
│           │       │   └── form.jsp
│           │       ├── user/
│           │       │   ├── list.jsp
│           │       │   ├── form.jsp
│           │       │   └── profile.jsp
│           │       └── dashboard.jsp
│           ├── static/
│           │   ├── css/
│           │   │   ├── style.css
│           │   │   └── bootstrap.min.css
│           │   ├── js/
│           │   │   ├── script.js
│           │   │   ├── jquery.min.js
│           │   │   └── bootstrap.min.js
│           │   └── img/
│           │       └── logo.png
│           ├── uploads/
│           │   └── .gitkeep
│           ├── error/
│           │   ├── 404.jsp
│           │   ├── 500.jsp
│           │   └── access-denied.jsp
│           └── index.jsp
└── pom.xml
```

3. Maven Configuration (pom.xml)

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>task-manager</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
    </properties>

    <dependencies>
        <!-- Servlet API -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>

        <!-- JSP API -->
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.3</version>
            <scope>provided</scope>
        </dependency>

        <!-- JSTL -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- MySQL Connector -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.28</version>
        </dependency>

        <!-- HikariCP Connection Pool -->
        <dependency>
            <groupId>com.zaxxer</groupId>
            <artifactId>HikariCP</artifactId>
            <version>5.0.1</version>
        </dependency>

        <!-- Jackson for JSON -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.13.4.2</version>
        </dependency>

        <!-- Bean Validation API -->
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>2.0.1.Final</version>
        </dependency>
```

```
        <!-- Hibernate Validator -->
        <dependency>
            <groupId>org.hibernate.validator</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>6.2.0.Final</version>
        </dependency>

        <!-- Apache Commons -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.12.0</version>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.11.0</version>
        </dependency>

        <!-- Log4j -->
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.17.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.17.1</version>
        </dependency>

        <!-- jBCrypt for password hashing -->
        <dependency>
            <groupId>org.mindrot</groupId>
            <artifactId>jbcrypt</artifactId>
            <version>0.4</version>
        </dependency>

        <!-- Testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>4.3.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
```

```
        <finalName>task-manager</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.2</version>
            </plugin>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <url>http://localhost:8080/manager/text</url>
                    <server>TomcatServer</server>
                    <path>/task-manager</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

6.3 Core Components Implementation

Let's implement the core components of our application, starting with configuration, models, and data access.

1. Database Configuration

DBConnectionPool.java:

```
package com.taskmanager.config;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;
import javax.sql.DataSource;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class DBConnectionPool {

    private static final Logger logger =
LogManager.getLogger(DBConnectionPool.class);
    private static HikariDataSource dataSource;

    static {
        try {
            Properties props = new Properties();
```

```
            InputStream inputStream = DBConnectionPool.class.getClassLoader()
                    .getResourceAsStream("db.properties");

            if (inputStream != null) {
                props.load(inputStream);

                HikariConfig config = new HikariConfig();
                config.setJdbcUrl(props.getProperty("db.url"));
                config.setUsername(props.getProperty("db.username"));
                config.setPassword(props.getProperty("db.password"));
                config.setDriverClassName(props.getProperty("db.driver"));

                // Connection pool settings

config.setMaximumPoolSize(Integer.parseInt(props.getProperty("db.maxPoolSize",
"10")));

config.setMinimumIdle(Integer.parseInt(props.getProperty("db.minIdle", "5")));

config.setIdleTimeout(Long.parseLong(props.getProperty("db.idleTimeout",
"30000")));

config.setConnectionTimeout(Long.parseLong(props.getProperty("db.connectionTimeout
", "30000")));

config.setMaxLifetime(Long.parseLong(props.getProperty("db.maxLifetime",
"1800000")));

                // Set pool name
                config.setPoolName("TaskManagerHikariCP");

                // Connection test query
                config.setConnectionTestQuery("SELECT 1");

                // Create the data source
                dataSource = new HikariDataSource(config);

                logger.info("Database connection pool initialized successfully");
            } else {
                logger.error("Could not find db.properties file");
                throw new RuntimeException("Could not find db.properties file");
            }
        } catch (IOException e) {
            logger.error("Error loading database properties", e);
            throw new RuntimeException("Error loading database properties", e);
        }
    }

    public static Connection getConnection() throws SQLException {
        if (dataSource == null) {
            throw new SQLException("Data source is not configured");
        }
        return dataSource.getConnection();
    }
```

```
    public static DataSource getDataSource() {
        return dataSource;
    }

    public static void closeDataSource() {
        if (dataSource != null && !dataSource.isClosed()) {
            dataSource.close();
            logger.info("Database connection pool closed");
        }
    }

    public static String getPoolStats() {
        if (dataSource != null) {
            return String.format(
                "Pool Stats: Active=%d, Idle=%d, Total=%d, Waiting=%d",
                dataSource.getHikariPoolMXBean().getActiveConnections(),
                dataSource.getHikariPoolMXBean().getIdleConnections(),
                dataSource.getHikariPoolMXBean().getTotalConnections(),
                dataSource.getHikariPoolMXBean().getThreadsAwaitingConnection()
            );
        }
        return "Pool not initialized";
    }
}
```

db.properties:

```
db.driver=com.mysql.cj.jdbc.Driver
db.url=jdbc:mysql://localhost:3306/task_manager?useSSL=false&serverTimezone=UTC
db.username=root
db.password=password
db.maxPoolSize=10
db.minIdle=5
db.idleTimeout=30000
db.connectionTimeout=30000
db.maxLifetime=1800000
```

2. Model Classes

User.java:

```
package com.taskmanager.model;

import java.util.Date;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;
```

```
public class User {

    private Long id;

    @NotBlank(message = "Username is required")
    @Size(min = 3, max = 50, message = "Username must be between 3 and 50
characters")
    private String username;

    @NotBlank(message = "Email is required")
    @Email(message = "Email must be valid")
    private String email;

    @NotBlank(message = "Password is required")
    @Size(min = 6, message = "Password must be at least 6 characters")
    private String password;

    @NotBlank(message = "Full name is required")
    private String fullName;

    private String role;
    private Date createdAt;
    private Date updatedAt;

    // Constructors
    public User() {
    }

    public User(Long id, String username, String email, String fullName, String
role) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.fullName = fullName;
        this.role = role;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
```

```
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }

    public Date getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(Date updatedAt) {
        this.updatedAt = updatedAt;
    }

    // Helper methods
    public boolean isAdmin() {
        return "ADMIN".equals(role);
    }

    @Override
```

```
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", email='" + email + '\'' +
                ", fullName='" + fullName + '\'' +
                ", role='" + role + '\'' +
                '}';
    }
}
```

Task.java:

```
package com.taskmanager.model;

import java.util.Date;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class Task {

    private Long id;

    @NotBlank(message = "Title is required")
    @Size(max = 100, message = "Title must not exceed 100 characters")
    private String title;

    private String description;
    private Date dueDate;

    @NotNull(message = "Priority is required")
    private String priority;

    @NotNull(message = "Status is required")
    private String status;

    private Long categoryId;
    private Category category;

    @NotNull(message = "Creator is required")
    private Long creatorId;
    private User creator;

    private Long assigneeId;
    private User assignee;

    private Date createdAt;
    private Date updatedAt;

    // Constructors
    public Task() {
```

```
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Date getDueDate() {
        return dueDate;
    }

    public void setDueDate(Date dueDate) {
        this.dueDate = dueDate;
    }

    public String getPriority() {
        return priority;
    }

    public void setPriority(String priority) {
        this.priority = priority;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public Long getCategoryId() {
        return categoryId;
    }
```

```
    public void setCategoryId(Long categoryId) {
        this.categoryId = categoryId;
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
        if (category != null) {
            this.categoryId = category.getId();
        }
    }

    public Long getCreatorId() {
        return creatorId;
    }

    public void setCreatorId(Long creatorId) {
        this.creatorId = creatorId;
    }

    public User getCreator() {
        return creator;
    }

    public void setCreator(User creator) {
        this.creator = creator;
        if (creator != null) {
            this.creatorId = creator.getId();
        }
    }

    public Long getAssigneeId() {
        return assigneeId;
    }

    public void setAssigneeId(Long assigneeId) {
        this.assigneeId = assigneeId;
    }

    public User getAssignee() {
        return assignee;
    }

    public void setAssignee(User assignee) {
        this.assignee = assignee;
        if (assignee != null) {
            this.assigneeId = assignee.getId();
        }
    }
```

```
    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }

    public Date getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(Date updatedAt) {
        this.updatedAt = updatedAt;
    }

    // Helper methods
    public boolean isOverdue() {
        if (dueDate == null || "DONE".equals(status)) {
            return false;
        }
        return dueDate.before(new Date());
    }

    public String getPriorityClass() {
        switch (priority) {
            case "HIGH": return "danger";
            case "MEDIUM": return "warning";
            case "LOW": return "info";
            default: return "secondary";
        }
    }

    public String getStatusClass() {
        switch (status) {
            case "TODO": return "secondary";
            case "IN_PROGRESS": return "primary";
            case "DONE": return "success";
            default: return "light";
        }
    }

    @Override
    public String toString() {
        return "Task{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", status='" + status + '\'' +
                ", priority='" + priority + '\'' +
                ", dueDate=" + dueDate +
                '}';
    }
}
```

Category.java:

```
package com.taskmanager.model;

import java.util.Date;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;

public class Category {

    private Long id;

    @NotBlank(message = "Name is required")
    private String name;

    @NotBlank(message = "Color is required")
    @Pattern(regexp = "^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$", message = "Color must
be a valid hex color")
    private String color;

    private Date createdAt;

    // Constructors
    public Category() {
    }

    public Category(Long id, String name, String color) {
        this.id = id;
        this.name = name;
        this.color = color;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getColor() {
        return color;
    }
```

```
    public void setColor(String color) {
        this.color = color;
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }

    @Override
    public String toString() {
        return "Category{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", color='" + color + '\'' +
                '}';
    }
}
```

3. Data Access Objects (DAOs)

UserDAO.java (Interface):

```
package com.taskmanager.dao;

import com.taskmanager.exception.DAOException;
import com.taskmanager.model.User;
import java.util.List;

public interface UserDAO {

    User findById(Long id) throws DAOException;

    User findByUsername(String username) throws DAOException;

    User findByEmail(String email) throws DAOException;

    List<User> findAll() throws DAOException;

    List<User> findByRole(String role) throws DAOException;

    Long save(User user) throws DAOException;

    void update(User user) throws DAOException;

    void delete(Long id) throws DAOException;
```

```
    boolean isUsernameExists(String username) throws DAOException;

    boolean isEmailExists(String email) throws DAOException;
}
```

JdbcUserDAO.java (Implementation):

```
package com.taskmanager.dao.impl;

import com.taskmanager.config.DBConnectionPool;
import com.taskmanager.dao.UserDAO;
import com.taskmanager.exception.DAOException;
import com.taskmanager.model.User;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class JdbcUserDAO implements UserDAO {

    private static final Logger logger = LogManager.getLogger(JdbcUserDAO.class);

    private static final String FIND_BY_ID_SQL =
            "SELECT * FROM users WHERE id = ?";

    private static final String FIND_BY_USERNAME_SQL =
            "SELECT * FROM users WHERE username = ?";

    private static final String FIND_BY_EMAIL_SQL =
            "SELECT * FROM users WHERE email = ?";

    private static final String FIND_ALL_SQL =
            "SELECT * FROM users ORDER BY username";

    private static final String FIND_BY_ROLE_SQL =
            "SELECT * FROM users WHERE role = ? ORDER BY username";

    private static final String INSERT_SQL =
            "INSERT INTO users (username, email, password_hash, full_name, role) "
+
            "VALUES (?, ?, ?, ?, ?)";

    private static final String UPDATE_SQL =
            "UPDATE users SET username = ?, email = ?, full_name = ?, role = ?, "
+
            "updated_at = CURRENT_TIMESTAMP WHERE id = ?";
```

```
    private static final String UPDATE_WITH_PASSWORD_SQL =
            "UPDATE users SET username = ?, email = ?, password_hash = ?,
full_name = ?, " +
            "role = ?, updated_at = CURRENT_TIMESTAMP WHERE id = ?";

    private static final String DELETE_SQL =
            "DELETE FROM users WHERE id = ?";

    private static final String CHECK_USERNAME_SQL =
            "SELECT COUNT(*) FROM users WHERE username = ?";

    private static final String CHECK_EMAIL_SQL =
            "SELECT COUNT(*) FROM users WHERE email = ?";

    @Override
    public User findById(Long id) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(FIND_BY_ID_SQL)) {

            stmt.setLong(1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return mapResultSetToUser(rs);
                }
                return null;
            }
        } catch (SQLException e) {
            logger.error("Error finding user by ID: " + id, e);
            throw new DAOException("Error finding user by ID: " + id, e);
        }
    }

    @Override
    public User findByUsername(String username) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(FIND_BY_USERNAME_SQL))
{

            stmt.setString(1, username);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return mapResultSetToUser(rs);
                }
                return null;
            }
        } catch (SQLException e) {
            logger.error("Error finding user by username: " + username, e);
            throw new DAOException("Error finding user by username: " + username,
e);
        }
    }

    @Override
```

```
    public User findByEmail(String email) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(FIND_BY_EMAIL_SQL)) {

            stmt.setString(1, email);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return mapResultSetToUser(rs);
                }
                return null;
            }
        } catch (SQLException e) {
            logger.error("Error finding user by email: " + email, e);
            throw new DAOException("Error finding user by email: " + email, e);
        }
    }

    @Override
    public List<User> findAll() throws DAOException {
        List<User> users = new ArrayList<>();
        try (Connection conn = DBConnectionPool.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(FIND_ALL_SQL)) {

            while (rs.next()) {
                users.add(mapResultSetToUser(rs));
            }
            return users;
        } catch (SQLException e) {
            logger.error("Error finding all users", e);
            throw new DAOException("Error finding all users", e);
        }
    }

    @Override
    public List<User> findByRole(String role) throws DAOException {
        List<User> users = new ArrayList<>();
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(FIND_BY_ROLE_SQL)) {

            stmt.setString(1, role);
            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    users.add(mapResultSetToUser(rs));
                }
                return users;
            }
        } catch (SQLException e) {
            logger.error("Error finding users by role: " + role, e);
            throw new DAOException("Error finding users by role: " + role, e);
        }
    }

    @Override
```

```
    public Long save(User user) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(INSERT_SQL,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, user.getUsername());
            stmt.setString(2, user.getEmail());
            stmt.setString(3, user.getPassword());
            stmt.setString(4, user.getFullName());
            stmt.setString(5, user.getRole() != null ? user.getRole() : "USER");

            int affectedRows = stmt.executeUpdate();

            if (affectedRows == 0) {
                throw new DAOException("Creating user failed, no rows affected.");
            }

            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    Long id = generatedKeys.getLong(1);
                    user.setId(id);
                    return id;
                } else {
                    throw new DAOException("Creating user failed, no ID
obtained.");
                }
            }
        } catch (SQLException e) {
            logger.error("Error saving user: " + user.getUsername(), e);
            throw new DAOException("Error saving user: " + user.getUsername(), e);
        }
    }

    @Override
    public void update(User user) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection()) {
            PreparedStatement stmt;

            if (user.getPassword() != null && !user.getPassword().isEmpty()) {
                stmt = conn.prepareStatement(UPDATE_WITH_PASSWORD_SQL);
                stmt.setString(1, user.getUsername());
                stmt.setString(2, user.getEmail());
                stmt.setString(3, user.getPassword());
                stmt.setString(4, user.getFullName());
                stmt.setString(5, user.getRole());
                stmt.setLong(6, user.getId());
            } else {
                stmt = conn.prepareStatement(UPDATE_SQL);
                stmt.setString(1, user.getUsername());
                stmt.setString(2, user.getEmail());
                stmt.setString(3, user.getFullName());
                stmt.setString(4, user.getRole());
                stmt.setLong(5, user.getId());
            }
```

```
            try {
                int affectedRows = stmt.executeUpdate();

                if (affectedRows == 0) {
                    throw new DAOException("Updating user failed, no rows
affected.");
                }
            } finally {
                stmt.close();
            }
        } catch (SQLException e) {
            logger.error("Error updating user: " + user.getId(), e);
            throw new DAOException("Error updating user: " + user.getId(), e);
        }
    }

    @Override
    public void delete(Long id) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(DELETE_SQL)) {

            stmt.setLong(1, id);

            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                throw new DAOException("Deleting user failed, no rows affected.");
            }
        } catch (SQLException e) {
            logger.error("Error deleting user: " + id, e);
            throw new DAOException("Error deleting user: " + id, e);
        }
    }

    @Override
    public boolean isUsernameExists(String username) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(CHECK_USERNAME_SQL)) {

            stmt.setString(1, username);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt(1) > 0;
                }
                return false;
            }
        } catch (SQLException e) {
            logger.error("Error checking username existence: " + username, e);
            throw new DAOException("Error checking username existence: " +
username, e);
        }
    }

    @Override
```

```
    public boolean isEmailExists(String email) throws DAOException {
        try (Connection conn = DBConnectionPool.getConnection();
             PreparedStatement stmt = conn.prepareStatement(CHECK_EMAIL_SQL)) {

            stmt.setString(1, email);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt(1) > 0;
                }
                return false;
            }
        } catch (SQLException e) {
            logger.error("Error checking email existence: " + email, e);
            throw new DAOException("Error checking email existence: " + email, e);
        }
    }

    private User mapResultSetToUser(ResultSet rs) throws SQLException {
        User user = new User();
        user.setId(rs.getLong("id"));
        user.setUsername(rs.getString("username"));
        user.setEmail(rs.getString("email"));
        user.setPassword(rs.getString("password_hash"));
        user.setFullName(rs.getString("full_name"));
        user.setRole(rs.getString("role"));

        Timestamp createdAt = rs.getTimestamp("created_at");
        if (createdAt != null) {
            user.setCreatedAt(new java.util.Date(createdAt.getTime()));
        }

        Timestamp updatedAt = rs.getTimestamp("updated_at");
        if (updatedAt != null) {
            user.setUpdatedAt(new java.util.Date(updatedAt.getTime()));
        }

        return user;
    }
}
```

TaskDAO.java and JdbcTaskDAO.java:

Similar to the UserDAO implementation, with methods for CRUD operations on tasks, including:

- Finding tasks by various criteria (ID, creator, assignee, status, etc.)
- Saving, updating, and deleting tasks
- Handling task-related queries (e.g., finding overdue tasks)

CategoryDAO.java and JdbcCategoryDAO.java:

Similar to other DAOs, implementing CRUD operations for categories.

4. Service Layer

UserService.java:

```
package com.taskmanager.service;

import com.taskmanager.dao.UserDAO;
import com.taskmanager.dao.impl.JdbcUserDAO;
import com.taskmanager.exception.DAOException;
import com.taskmanager.exception.ServiceException;
import com.taskmanager.exception.ValidationException;
import com.taskmanager.model.User;
import com.taskmanager.util.PasswordUtil;
import com.taskmanager.util.ValidationUtil;
import java.util.List;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class UserService {

    private static final Logger logger = LogManager.getLogger(UserService.class);
    private UserDAO userDAO;

    public UserService() {
        this.userDAO = new JdbcUserDAO();
    }

    public UserService(UserDAO userDAO) {
        this.userDAO = userDAO;
    }

    public User findById(Long id) throws ServiceException {
        try {
            return userDAO.findById(id);
        } catch (DAOException e) {
            logger.error("Error finding user by ID: " + id, e);
            throw new ServiceException("Error finding user by ID: " + id, e);
        }
    }

    public User findByUsername(String username) throws ServiceException {
        try {
            return userDAO.findByUsername(username);
        } catch (DAOException e) {
            logger.error("Error finding user by username: " + username, e);
            throw new ServiceException("Error finding user by username: " +
username, e);
        }
    }

    public List<User> findAll() throws ServiceException {
        try {
```

```
            return userDAO.findAll();
        } catch (DAOException e) {
            logger.error("Error finding all users", e);
            throw new ServiceException("Error finding all users", e);
        }
    }

    public List<User> findByRole(String role) throws ServiceException {
        try {
            return userDAO.findByRole(role);
        } catch (DAOException e) {
            logger.error("Error finding users by role: " + role, e);
            throw new ServiceException("Error finding users by role: " + role, e);
        }
    }

    public Long registerUser(User user) throws ServiceException,
ValidationException {
        try {
            // Validate user data
            ValidationUtil.validateUser(user);

            // Check if username or email already exists
            if (userDAO.isUsernameExists(user.getUsername())) {
                throw new ValidationException("Username already exists");
            }

            if (userDAO.isEmailExists(user.getEmail())) {
                throw new ValidationException("Email already exists");
            }

            // Hash the password
            String hashedPassword = PasswordUtil.hashPassword(user.getPassword());
            user.setPassword(hashedPassword);

            // Set default role if not provided
            if (user.getRole() == null || user.getRole().isEmpty()) {
                user.setRole("USER");
            }

            // Save the user
            return userDAO.save(user);

        } catch (DAOException e) {
            logger.error("Error registering user: " + user.getUsername(), e);
            throw new ServiceException("Error registering user: " +
user.getUsername(), e);
        }
    }

    public void updateUser(User user) throws ServiceException, ValidationException
{
        try {
            // Validate user data
```

```
            ValidationUtil.validateUserUpdate(user);

            // Get existing user
            User existingUser = userDAO.findById(user.getId());
            if (existingUser == null) {
                throw new ValidationException("User not found");
            }

            // Check if username is changed and already exists
            if (!existingUser.getUsername().equals(user.getUsername()) &&
                    userDAO.isUsernameExists(user.getUsername())) {
                throw new ValidationException("Username already exists");
            }

            // Check if email is changed and already exists
            if (!existingUser.getEmail().equals(user.getEmail()) &&
                    userDAO.isEmailExists(user.getEmail())) {
                throw new ValidationException("Email already exists");
            }

            // Hash the password if provided
            if (user.getPassword() != null && !user.getPassword().isEmpty()) {
                String hashedPassword =
PasswordUtil.hashPassword(user.getPassword());
                user.setPassword(hashedPassword);
            } else {
                user.setPassword(null); // Don't update password
            }

            // Update the user
            userDAO.update(user);

        } catch (DAOException e) {
            logger.error("Error updating user: " + user.getId(), e);
            throw new ServiceException("Error updating user: " + user.getId(), e);
        }
    }

    public void deleteUser(Long id) throws ServiceException {
        try {
            userDAO.delete(id);
        } catch (DAOException e) {
            logger.error("Error deleting user: " + id, e);
            throw new ServiceException("Error deleting user: " + id, e);
        }
    }

    public User authenticateUser(String username, String password) throws
ServiceException {
        try {
            User user = userDAO.findByUsername(username);
            if (user != null && PasswordUtil.checkPassword(password,
user.getPassword())) {
                return user;
```

```
            }
            return null;
        } catch (DAOException e) {
            logger.error("Error authenticating user: " + username, e);
            throw new ServiceException("Error authenticating user: " + username,
e);
        }
    }
}
```

TaskService.java and CategoryService.java:

Similar to UserService, implementing business logic for tasks and categories.

AuthService.java:

```
package com.taskmanager.service;

import com.taskmanager.exception.AuthenticationException;
import com.taskmanager.exception.ServiceException;
import com.taskmanager.model.User;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class AuthService {

    private static final Logger logger = LogManager.getLogger(AuthService.class);
    private static final String USER_SESSION_ATTRIBUTE = "user";

    private UserService userService;

    public AuthService() {
        this.userService = new UserService();
    }

    public AuthService(UserService userService) {
        this.userService = userService;
    }

    public User login(String username, String password) throws
AuthenticationException {
        try {
            User user = userService.authenticateUser(username, password);
            if (user == null) {
                throw new AuthenticationException("Invalid username or password");
            }
            return user;
        } catch (ServiceException e) {
            logger.error("Error during login for user: " + username, e);
            throw new AuthenticationException("Error during login", e);
```

```
        }
    }

    public void storeUserInSession(HttpServletRequest request, User user) {
        HttpSession session = request.getSession();
        session.setAttribute(USER_SESSION_ATTRIBUTE, user);
        logger.info("User stored in session: " + user.getUsername());
    }

    public void logout(HttpServletRequest request) {
        HttpSession session = request.getSession(false);
        if (session != null) {
            User user = (User) session.getAttribute(USER_SESSION_ATTRIBUTE);
            if (user != null) {
                logger.info("User logged out: " + user.getUsername());
            }
            session.invalidate();
        }
    }

    public User getCurrentUser(HttpServletRequest request) {
        HttpSession session = request.getSession(false);
        if (session != null) {
            return (User) session.getAttribute(USER_SESSION_ATTRIBUTE);
        }
        return null;
    }

    public boolean isAuthenticated(HttpServletRequest request) {
        return getCurrentUser(request) != null;
    }

    public boolean hasRole(HttpServletRequest request, String role) {
        User user = getCurrentUser(request);
        return user != null && role.equals(user.getRole());
    }

    public boolean isAdmin(HttpServletRequest request) {
        return hasRole(request, "ADMIN");
    }
}
```

5. Utility Classes

PasswordUtil.java:

```
package com.taskmanager.util;

import org.mindrot.jbcrypt.BCrypt;

public class PasswordUtil {
```

```
    private static final int BCRYPT_ROUNDS = 12;

    public static String hashPassword(String plainTextPassword) {
        return BCrypt.hashpw(plainTextPassword, BCrypt.gensalt(BCRYPT_ROUNDS));
    }

    public static boolean checkPassword(String plainTextPassword, String
hashedPassword) {
        return BCrypt.checkpw(plainTextPassword, hashedPassword);
    }
}
```

ValidationUtil.java:

```
package com.taskmanager.util;

import com.taskmanager.exception.ValidationException;
import com.taskmanager.model.Category;
import com.taskmanager.model.Task;
import com.taskmanager.model.User;
import java.util.HashSet;
import java.util.Set;
import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;

public class ValidationUtil {

    private static final ValidatorFactory factory =
Validation.buildDefaultValidatorFactory();
    private static final Validator validator = factory.getValidator();

    public static void validateUser(User user) throws ValidationException {
        validate(user);
    }

    public static void validateUserUpdate(User user) throws ValidationException {
        // Skip password validation for updates if not provided
        if (user.getPassword() == null || user.getPassword().isEmpty()) {
            // Custom validation without password
            if (user.getUsername() == null || user.getUsername().trim().isEmpty())
{
                throw new ValidationException("Username is required");
            }

            if (user.getEmail() == null || user.getEmail().trim().isEmpty()) {
                throw new ValidationException("Email is required");
            }

            if (user.getFullName() == null || user.getFullName().trim().isEmpty())
```

```
{
                throw new ValidationException("Full name is required");
            }
        } else {
            validate(user);
        }
    }

    public static void validateTask(Task task) throws ValidationException {
        validate(task);
    }

    public static void validateCategory(Category category) throws
ValidationException {
        validate(category);
    }

    private static <T> void validate(T object) throws ValidationException {
        Set<ConstraintViolation<T>> violations = validator.validate(object);

        if (!violations.isEmpty()) {
            Set<String> errorMessages = new HashSet<>();
            for (ConstraintViolation<T> violation : violations) {
                errorMessages.add(violation.getMessage());
            }
            throw new ValidationException(String.join(", ", errorMessages));
        }
    }
}
```

6.4 Web Layer Implementation

1. Filters

AuthenticationFilter.java:

```
package com.taskmanager.web.filter;

import com.taskmanager.model.User;
import com.taskmanager.service.AuthService;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebFilter;
import org.apache.logging.log4j.LogManager;
```

```
import org.apache.logging.log4j.Logger;

@WebFilter("/*")
public class AuthenticationFilter implements Filter {

    private static final Logger logger =
LogManager.getLogger(AuthenticationFilter.class);

    private AuthService authService;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        authService = new AuthService();
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        String requestURI = httpRequest.getRequestURI();

        // Allow access to login page, static resources, and API endpoints
        if (isPublicResource(requestURI)) {
            chain.doFilter(request, response);
            return;
        }

        // Check if user is authenticated
        User user = authService.getCurrentUser(httpRequest);

        if (user == null) {
            // User is not authenticated, redirect to login page
            logger.info("Unauthenticated access attempt to " + requestURI);
            httpResponse.sendRedirect(httpRequest.getContextPath() + "/login");
            return;
        }

        // Check admin-only areas
        if (isAdminOnlyResource(requestURI) && !authService.isAdmin(httpRequest))
{
            logger.warn("Unauthorized access attempt to admin area: " + requestURI
+
                    " by user: " + user.getUsername());
            httpResponse.sendRedirect(httpRequest.getContextPath() + "/access-
denied");
            return;
        }

        // User is authenticated and authorized, proceed
        chain.doFilter(request, response);
```

```
    }

    @Override
    public void destroy() {
        // Clean up resources
    }

    private boolean isPublicResource(String uri) {
        return uri.endsWith("/login") ||
               uri.endsWith("/register") ||
               uri.endsWith("/logout") ||
               uri.contains("/api/") ||
               uri.contains("/static/") ||
               uri.contains("/css/") ||
               uri.contains("/js/") ||
               uri.contains("/img/") ||
               uri.endsWith(".ico");
    }

    private boolean isAdminOnlyResource(String uri) {
        return uri.contains("/admin/") ||
               uri.contains("/users/") && !uri.contains("/profile");
    }
}
```

LoggingFilter.java:

```
package com.taskmanager.web.filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

@WebFilter("/*")
public class LoggingFilter implements Filter {

    private static final Logger logger =
LogManager.getLogger(LoggingFilter.class);

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Initialization code
    }
```

```
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        long startTime = System.currentTimeMillis();

        String requestURI = httpRequest.getRequestURI();
        String method = httpRequest.getMethod();
        String userAgent = httpRequest.getHeader("User-Agent");
        String remoteAddr = httpRequest.getRemoteAddr();

        logger.info("Request received: {} {} from {} ({})",
                method, requestURI, remoteAddr, userAgent);

        try {
            chain.doFilter(request, response);
        } finally {
            long endTime = System.currentTimeMillis();
            long duration = endTime - startTime;

            int status = httpResponse.getStatus();

            logger.info("Request completed: {} {} - {} ({} ms)",
                    method, requestURI, status, duration);
        }
    }

    @Override
    public void destroy() {
        // Cleanup code
    }
}
```

2. Servlets

LoginServlet.java:

```
package com.taskmanager.web.servlet.auth;

import com.taskmanager.exception.AuthenticationException;
import com.taskmanager.model.User;
import com.taskmanager.service.AuthService;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

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

    private static final Logger logger = LogManager.getLogger(LoginServlet.class);

    private AuthService authService;

    @Override
    public void init() throws ServletException {
        authService = new AuthService();
    }

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

        // If user is already authenticated, redirect to tasks page
        if (authService.isAuthenticated(request)) {
            response.sendRedirect(request.getContextPath() + "/tasks");
            return;
        }

        // Forward to login page
        request.getRequestDispatcher("/WEB-
INF/views/auth/login.jsp").forward(request, response);
    }

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

        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String redirectPath = request.getParameter("redirect");

        if (redirectPath == null || redirectPath.trim().isEmpty()) {
            redirectPath = request.getContextPath() + "/tasks";
        }

        try {
            // Attempt to authenticate
            User user = authService.login(username, password);

            // Store user in session
            authService.storeUserInSession(request, user);

            // Redirect to tasks page or requested URL
            response.sendRedirect(redirectPath);
```

```
        } catch (AuthenticationException e) {
            logger.info("Failed login attempt for username: " + username);

            // Add error message and forward back to login page
            request.setAttribute("error", e.getMessage());
            request.setAttribute("username", username);
            request.getRequestDispatcher("/WEB-
INF/views/auth/login.jsp").forward(request, response);
        }
    }
}
```

TaskListServlet.java:

```
package com.taskmanager.web.servlet.task;

import com.taskmanager.model.Category;
import com.taskmanager.model.Task;
import com.taskmanager.model.User;
import com.taskmanager.service.AuthService;
import com.taskmanager.service.CategoryService;
import com.taskmanager.service.TaskService;
import java.io.IOException;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

@WebServlet("/tasks")
public class TaskListServlet extends HttpServlet {

    private static final Logger logger =
LogManager.getLogger(TaskListServlet.class);

    private TaskService taskService;
    private CategoryService categoryService;
    private AuthService authService;

    @Override
    public void init() throws ServletException {
        taskService = new TaskService();
        categoryService = new CategoryService();
        authService = new AuthService();
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
            throws ServletException, IOException {

        User currentUser = authService.getCurrentUser(request);

        try {
            // Get filter parameters
            String status = request.getParameter("status");
            String categoryId = request.getParameter("categoryId");
            String priority = request.getParameter("priority");
            String searchTerm = request.getParameter("search");

            // Get tasks based on filters
            List<Task> tasks;

            if (authService.isAdmin(request)) {
                // Admins can see all tasks
                tasks = taskService.findAllWithFilters(status, categoryId,
priority, searchTerm);
            } else {
                // Regular users see only their tasks (created or assigned)
                tasks = taskService.findByUserWithFilters(currentUser.getId(),
status, categoryId, priority, searchTerm);
            }

            // Get categories for filter dropdown
            List<Category> categories = categoryService.findAll();

            // Set attributes
            request.setAttribute("tasks", tasks);
            request.setAttribute("categories", categories);
            request.setAttribute("selectedStatus", status);
            request.setAttribute("selectedCategoryId", categoryId);
            request.setAttribute("selectedPriority", priority);
            request.setAttribute("searchTerm", searchTerm);

            // Forward to task list view
            request.getRequestDispatcher("/WEB-
INF/views/task/list.jsp").forward(request, response);

        } catch (Exception e) {
            logger.error("Error retrieving tasks", e);
            request.setAttribute("error", "An error occurred while retrieving
tasks: " + e.getMessage());
            request.getRequestDispatcher("/WEB-
INF/views/task/list.jsp").forward(request, response);
        }
    }
}
```

TaskFormServlet.java:

```
package com.taskmanager.web.servlet.task;

import com.taskmanager.exception.ServiceException;
import com.taskmanager.exception.ValidationException;
import com.taskmanager.model.Category;
import com.taskmanager.model.Task;
import com.taskmanager.model.User;
import com.taskmanager.service.AuthService;
import com.taskmanager.service.CategoryService;
import com.taskmanager.service.TaskService;
import com.taskmanager.service.UserService;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

@WebServlet({"/tasks/add", "/tasks/edit/*"})
public class TaskFormServlet extends HttpServlet {

    private static final Logger logger =
LogManager.getLogger(TaskFormServlet.class);

    private TaskService taskService;
    private CategoryService categoryService;
    private UserService userService;
    private AuthService authService;

    @Override
    public void init() throws ServletException {
        taskService = new TaskService();
        categoryService = new CategoryService();
        userService = new UserService();
        authService = new AuthService();
    }

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

        User currentUser = authService.getCurrentUser(request);

        try {
            // Get categories and users for dropdown lists
            List<Category> categories = categoryService.findAll();
            List<User> users = userService.findAll();
```

```
            request.setAttribute("categories", categories);
            request.setAttribute("users", users);

            // Check if editing or adding
            String uri = request.getRequestURI();
            if (uri.contains("/edit/")) {
                // Extract task ID from URL
                String pathInfo = request.getPathInfo();
                if (pathInfo != null && pathInfo.startsWith("/")) {
                    String idStr = pathInfo.substring(1);
                    Long taskId = Long.parseLong(idStr);

                    // Get task for editing
                    Task task = taskService.findById(taskId);

                    // Check if task exists
                    if (task == null) {
                        response.sendRedirect(request.getContextPath() +
"/tasks");
                        return;
                    }

                    // Check if user has permission to edit
                    boolean canEdit = authService.isAdmin(request) ||

task.getCreatorId().equals(currentUser.getId()) ||
                                      (task.getAssigneeId() != null &&
task.getAssigneeId().equals(currentUser.getId()));

                    if (!canEdit) {
                        response.sendRedirect(request.getContextPath() + "/access-
denied");
                        return;
                    }

                    request.setAttribute("task", task);
                    request.setAttribute("formTitle", "Edit Task");
                    request.setAttribute("submitAction", "update");
                } else {
                    response.sendRedirect(request.getContextPath() + "/tasks");
                    return;
                }
            } else {
                // Adding a new task
                Task newTask = new Task();
                newTask.setCreatorId(currentUser.getId());
                newTask.setStatus("TODO");
                newTask.setPriority("MEDIUM");

                request.setAttribute("task", newTask);
                request.setAttribute("formTitle", "Add New Task");
                request.setAttribute("submitAction", "create");
            }
```

```
            // Forward to form
            request.getRequestDispatcher("/WEB-
INF/views/task/form.jsp").forward(request, response);

        } catch (ServiceException e) {
            logger.error("Error preparing task form", e);
            request.setAttribute("error", "An error occurred: " + e.getMessage());
            response.sendRedirect(request.getContextPath() + "/tasks");
        } catch (NumberFormatException e) {
            logger.error("Invalid task ID format", e);
            response.sendRedirect(request.getContextPath() + "/tasks");
        }
    }

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

        User currentUser = authService.getCurrentUser(request);

        try {
            // Determine if creating or updating
            String action = request.getParameter("action");
            boolean isUpdate = "update".equals(action);

            // Get form parameters
            String taskIdStr = request.getParameter("id");
            String title = request.getParameter("title");
            String description = request.getParameter("description");
            String dueDateStr = request.getParameter("dueDate");
            String priority = request.getParameter("priority");
            String status = request.getParameter("status");
            String categoryIdStr = request.getParameter("categoryId");
            String assigneeIdStr = request.getParameter("assigneeId");

            // Create task object
            Task task = new Task();

            if (isUpdate && taskIdStr != null && !taskIdStr.isEmpty()) {
                task.setId(Long.parseLong(taskIdStr));

                // Check if task exists
                Task existingTask = taskService.findById(task.getId());
                if (existingTask == null) {
                    request.setAttribute("error", "Task not found.");
                    response.sendRedirect(request.getContextPath() + "/tasks");
                    return;
                }

                // Check if user has permission to edit
                boolean canEdit = authService.isAdmin(request) ||
```

```
existingTask.getCreatorId().equals(currentUser.getId()) ||
                                (existingTask.getAssigneeId() != null &&

existingTask.getAssigneeId().equals(currentUser.getId()));

                if (!canEdit) {
                    response.sendRedirect(request.getContextPath() + "/access-
denied");
                    return;
                }

                // Preserve creator ID
                task.setCreatorId(existingTask.getCreatorId());
            } else {
                // New task
                task.setCreatorId(currentUser.getId());
            }

            // Set task properties
            task.setTitle(title);
            task.setDescription(description);

            // Parse and set due date if provided
            if (dueDateStr != null && !dueDateStr.isEmpty()) {
                SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
                Date dueDate = dateFormat.parse(dueDateStr);
                task.setDueDate(dueDate);
            }

            task.setPriority(priority);
            task.setStatus(status);

            // Set category ID if provided
            if (categoryIdStr != null && !categoryIdStr.isEmpty()) {
                task.setCategoryId(Long.parseLong(categoryIdStr));
            }

            // Set assignee ID if provided
            if (assigneeIdStr != null && !assigneeIdStr.isEmpty()) {
                task.setAssigneeId(Long.parseLong(assigneeIdStr));
            }

            // Save or update task
            if (isUpdate) {
                taskService.updateTask(task);
                request.setAttribute("message", "Task updated successfully.");
            } else {
                taskService.createTask(task);
                request.setAttribute("message", "Task created successfully.");
            }

            // Redirect to task list
            response.sendRedirect(request.getContextPath() + "/tasks");
```

```
        } catch (ValidationException e) {
            logger.warn("Validation error: " + e.getMessage());
            request.setAttribute("error", e.getMessage());
            request.getRequestDispatcher("/WEB-
INF/views/task/form.jsp").forward(request, response);
        } catch (ServiceException e) {
            logger.error("Service error: " + e.getMessage(), e);
            request.setAttribute("error", "An error occurred: " + e.getMessage());
            request.getRequestDispatcher("/WEB-
INF/views/task/form.jsp").forward(request, response);
        } catch (ParseException e) {
            logger.warn("Date parsing error: " + e.getMessage());
            request.setAttribute("error", "Invalid date format: " +
e.getMessage());
            request.getRequestDispatcher("/WEB-
INF/views/task/form.jsp").forward(request, response);
        } catch (Exception e) {
            logger.error("Unexpected error: " + e.getMessage(), e);
            request.setAttribute("error", "An unexpected error occurred: " +
e.getMessage());
            request.getRequestDispatcher("/WEB-
INF/views/task/form.jsp").forward(request, response);
        }
    }
}
```

Other servlets follow a similar pattern, handling their specific responsibilities.

3. Custom Tags

pagination.tag:

```
<%@ tag description="Pagination Tag" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ attribute name="currentPage" required="true" type="java.lang.Integer" %>
<%@ attribute name="totalPages" required="true" type="java.lang.Integer" %>
<%@ attribute name="baseUrl" required="true" type="java.lang.String" %>

<c:if test="${totalPages > 1}">
    <nav aria-label="Page navigation">
        <ul class="pagination">
            <c:choose>
                <c:when test="${currentPage == 1}">
                    <li class="page-item disabled">
                        <span class="page-link">Previous</span>
                    </li>
                </c:when>
                <c:otherwise>
                    <li class="page-item">
                        <a class="page-link" href="${baseUrl}page=${currentPage -
```

```
1}">Previous</a>
                    </li>
                </c:otherwise>
            </c:choose>

            <c:forEach begin="1" end="${totalPages}" var="i">
                <c:choose>
                    <c:when test="${i == currentPage}">
                        <li class="page-item active">
                            <span class="page-link">${i}</span>
                        </li>
                    </c:when>
                    <c:otherwise>
                        <li class="page-item">
                            <a class="page-link" href="${baseUrl}page=${i}">${i}
</a>
                        </li>
                    </c:otherwise>
                </c:choose>
            </c:forEach>

            <c:choose>
                <c:when test="${currentPage == totalPages}">
                    <li class="page-item disabled">
                        <span class="page-link">Next</span>
                    </li>
                </c:when>
                <c:otherwise>
                    <li class="page-item">
                        <a class="page-link" href="${baseUrl}page=${currentPage +
1}">Next</a>
                    </li>
                </c:otherwise>
            </c:choose>
        </ul>
    </nav>
</c:if>
```

formatDate.tag:

```
<%@ tag description="Format Date Tag" pageEncoding="UTF-8" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ attribute name="date" required="true" type="java.util.Date" %>
<%@ attribute name="pattern" required="false" type="java.lang.String" %>
<%@ attribute name="relativeFormat" required="false" type="java.lang.Boolean" %>

<c:if test="${empty pattern}">
    <c:set var="pattern" value="yyyy-MM-dd" />
</c:if>
```

```
<c:if test="${empty relativeFormat}">
    <c:set var="relativeFormat" value="false" />
</c:if>

<c:if test="${date != null}">
    <c:choose>
        <c:when test="${relativeFormat}">
            <c:set var="now" value="<%= new java.util.Date() %>" />
            <c:set var="diffMs" value="${now.time - date.time}" />
            <c:set var="diffSec" value="${diffMs / 1000}" />
            <c:set var="diffMin" value="${diffSec / 60}" />
            <c:set var="diffHours" value="${diffMin / 60}" />
            <c:set var="diffDays" value="${diffHours / 24}" />

            <c:choose>
                <c:when test="${diffSec < 60}">
                    <span title="<fmt:formatDate value="${date}"
pattern="${pattern}" />">Just now</span>
                </c:when>
                <c:when test="${diffMin < 60}">
                    <span title="<fmt:formatDate value="${date}"
pattern="${pattern}" />">
                        ${Math.floor(diffMin)} minute<c:if
test="${Math.floor(diffMin) > 1}">s</c:if> ago
                    </span>
                </c:when>
                <c:when test="${diffHours < 24}">
                    <span title="<fmt:formatDate value="${date}"
pattern="${pattern}" />">
                        ${Math.floor(diffHours)} hour<c:if
test="${Math.floor(diffHours) > 1}">s</c:if> ago
                    </span>
                </c:when>
                <c:when test="${diffDays < 7}">
                    <span title="<fmt:formatDate value="${date}"
pattern="${pattern}" />">
                        ${Math.floor(diffDays)} day<c:if
test="${Math.floor(diffDays) > 1}">s</c:if> ago
                    </span>
                </c:when>
                <c:otherwise>
                    <fmt:formatDate value="${date}" pattern="${pattern}" />
                </c:otherwise>
            </c:choose>
        </c:when>
        <c:otherwise>
            <fmt:formatDate value="${date}" pattern="${pattern}" />
        </c:otherwise>
    </c:choose>
</c:if>
<c:if test="${date == null}">
    <span class="text-muted">Not set</span>
</c:if>
```

taskmanager.tld:

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd"
        version="2.1">

    <tlib-version>1.0</tlib-version>
    <short-name>tm</short-name>
    <uri>http://taskmanager.com/tags</uri>

    <tag-file>
        <name>pagination</name>
        <path>/WEB-INF/tags/pagination.tag</path>
    </tag-file>

    <tag-file>
        <name>formatDate</name>
        <path>/WEB-INF/tags/formatDate.tag</path>
    </tag-file>

    <tag>
        <name>hasRole</name>
        <tag-class>com.taskmanager.web.tag.HasRoleTag</tag-class>
        <body-content>JSP</body-content>
        <attribute>
            <name>role</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

6.5 JSP Views

Let's implement the JSP views for our application.

1. Common Templates

header.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="tm" uri="http://taskmanager.com/tags" %>

<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>${param.title} - Task Manager</title>

    <!-- Bootstrap CSS -->
    <link href="${pageContext.request.contextPath}/static/css/bootstrap.min.css"
rel="stylesheet">

    <!-- Custom CSS -->
    <link href="${pageContext.request.contextPath}/static/css/style.css"
rel="stylesheet">

    <!-- Font Awesome -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css">
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
            <div class="container">
                <a class="navbar-brand"
href="${pageContext.request.contextPath}/">
                    <img
src="${pageContext.request.contextPath}/static/img/logo.png" alt="Logo"
height="30" class="d-inline-block align-top">
                    Task Manager
                </a>

                <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav">
                    <span class="navbar-toggler-icon"></span>
                </button>

                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav me-auto">
                        <c:if test="${not empty sessionScope.user}">
                            <li class="nav-item">
                                <a class="nav-link ${param.activeNav ==
'dashboard' ? 'active' : ''}" href="${pageContext.request.contextPath}/dashboard">
                                    <i class="fas fa-tachometer-alt"></i>
Dashboard
                                </a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link ${param.activeNav == 'tasks' ?
'active' : ''}" href="${pageContext.request.contextPath}/tasks">
                                    <i class="fas fa-tasks"></i> Tasks
                                </a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link ${param.activeNav ==
'categories' ? 'active' : ''}"
href="${pageContext.request.contextPath}/categories">
```

```
                                    <i class="fas fa-tags"></i> Categories
                                </a>
                            </li>
                            <tm:hasRole role="ADMIN">
                                <li class="nav-item">
                                    <a class="nav-link ${param.activeNav ==
'users' ? 'active' : ''}" href="${pageContext.request.contextPath}/users">
                                        <i class="fas fa-users"></i> Users
                                    </a>
                                </li>
                            </tm:hasRole>
                        </c:if>
                    </ul>

                    <ul class="navbar-nav">
                        <c:choose>
                            <c:when test="${empty sessionScope.user}">
                                <li class="nav-item">
                                    <a class="nav-link ${param.activeNav ==
'login' ? 'active' : ''}" href="${pageContext.request.contextPath}/login">
                                        <i class="fas fa-sign-in-alt"></i> Login
                                    </a>
                                </li>
                                <li class="nav-item">
                                    <a class="nav-link ${param.activeNav ==
'register' ? 'active' : ''}" href="${pageContext.request.contextPath}/register">
                                        <i class="fas fa-user-plus"></i> Register
                                    </a>
                                </li>
                            </c:when>
                            <c:otherwise>
                                <li class="nav-item dropdown">
                                    <a class="nav-link dropdown-toggle" href="#"
id="userDropdown" role="button" data-bs-toggle="dropdown">
                                        <i class="fas fa-user-circle"></i>
${sessionScope.user.fullName}
                                    </a>
                                    <ul class="dropdown-menu dropdown-menu-end">
                                        <li>
                                            <a class="dropdown-item"
href="${pageContext.request.contextPath}/users/profile">
                                                <i class="fas fa-id-card"></i>
Profile
                                            </a>
                                        </li>
                                        <li><hr class="dropdown-divider"></li>
                                        <li>
                                            <a class="dropdown-item"
href="${pageContext.request.contextPath}/logout">
                                                <i class="fas fa-sign-out-alt">
</i> Logout
                                            </a>
                                        </li>
                                    </ul>
```

```
                                </li>
                            </c:otherwise>
                        </c:choose>
                    </ul>
                </div>
            </div>
        </nav>
    </header>

    <main class="container mt-4">
        <!-- Display alerts for messages and errors -->
        <c:if test="${not empty message}">
            <div class="alert alert-success alert-dismissible fade show"
role="alert">
                ${message}
                <button type="button" class="btn-close" data-bs-dismiss="alert"
aria-label="Close"></button>
            </div>
        </c:if>

        <c:if test="${not empty error}">
            <div class="alert alert-danger alert-dismissible fade show"
role="alert">
                ${error}
                <button type="button" class="btn-close" data-bs-dismiss="alert"
aria-label="Close"></button>
            </div>
        </c:if>
```

footer.jsp:

```
    </main>

    <footer class="bg-light py-3 mt-5">
        <div class="container text-center">
            <p class="mb-0">&copy; 2023 Task Manager Application</p>
        </div>
    </footer>

    <!-- jQuery -->
    <script src="${pageContext.request.contextPath}/static/js/jquery.min.js">
</script>

    <!-- Bootstrap JS Bundle with Popper -->
    <script src="${pageContext.request.contextPath}/static/js/bootstrap.min.js">
</script>

    <!-- Custom JavaScript -->
    <script src="${pageContext.request.contextPath}/static/js/script.js"></script>
</body>
</html>
```

2. Auth Views

login.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!-- Include Header -->
<jsp:include page="/WEB-INF/views/common/header.jsp">
    <jsp:param name="title" value="Login" />
    <jsp:param name="activeNav" value="login" />
</jsp:include>

<div class="row justify-content-center">
    <div class="col-md-6 col-lg-4">
        <div class="card shadow">
            <div class="card-header bg-primary text-white">
                <h4 class="mb-0"><i class="fas fa-sign-in-alt"></i> Login</h4>
            </div>
            <div class="card-body">
                <form action="${pageContext.request.contextPath}/login"
method="post">
                    <input type="hidden" name="redirect" value="${param.redirect}"
/>

                    <div class="mb-3">
                        <label for="username" class="form-label">Username</label>
                        <div class="input-group">
                            <span class="input-group-text"><i class="fas fa-user">
</i></span>
                            <input type="text" class="form-control" id="username"
name="username"
                                   value="${username}" required autofocus>
                        </div>
                    </div>

                    <div class="mb-3">
                        <label for="password" class="form-label">Password</label>
                        <div class="input-group">
                            <span class="input-group-text"><i class="fas fa-lock">
</i></span>
                            <input type="password" class="form-control"
id="password" name="password" required>
                        </div>
                    </div>

                    <div class="d-grid gap-2">
                        <button type="submit" class="btn btn-primary">
                            <i class="fas fa-sign-in-alt"></i> Login
                        </button>
                    </div>
```

```
                </form>
            </div>
            <div class="card-footer text-center">
                <p class="mb-0">Don't have an account? <a
href="${pageContext.request.contextPath}/register">Register</a></p>
            </div>
        </div>
    </div>
</div>

<!-- Include Footer -->
<jsp:include page="/WEB-INF/views/common/footer.jsp" />
```

register.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!-- Include Header -->
<jsp:include page="/WEB-INF/views/common/header.jsp">
    <jsp:param name="title" value="Register" />
    <jsp:param name="activeNav" value="register" />
</jsp:include>

<div class="row justify-content-center">
    <div class="col-md-8 col-lg-6">
        <div class="card shadow">
            <div class="card-header bg-primary text-white">
                <h4 class="mb-0"><i class="fas fa-user-plus"></i> Register</h4>
            </div>
            <div class="card-body">
                <form action="${pageContext.request.contextPath}/register"
method="post" id="registerForm">
                    <div class="row">
                        <div class="col-md-6 mb-3">
                            <label for="username" class="form-
label">Username</label>
                            <div class="input-group">
                                <span class="input-group-text"><i class="fas fa-
user"></i></span>
                                <input type="text" class="form-control"
id="username" name="username"
                                       value="${user.username}" required
minlength="3" maxlength="50">
                            </div>
                            <div class="form-text">Username must be 3-50
characters long.</div>
                        </div>

                        <div class="col-md-6 mb-3">
                            <label for="email" class="form-label">Email</label>
                            <div class="input-group">
```

```
                                <span class="input-group-text"><i class="fas fa-
envelope"></i></span>
                                <input type="email" class="form-control"
id="email" name="email"
                                       value="${user.email}" required>
                            </div>
                        </div>
                    </div>

                    <div class="mb-3">
                        <label for="fullName" class="form-label">Full Name</label>
                        <div class="input-group">
                            <span class="input-group-text"><i class="fas fa-id-
card"></i></span>
                            <input type="text" class="form-control" id="fullName"
name="fullName"
                                   value="${user.fullName}" required>
                        </div>
                    </div>

                    <div class="row">
                        <div class="col-md-6 mb-3">
                            <label for="password" class="form-
label">Password</label>
                            <div class="input-group">
                                <span class="input-group-text"><i class="fas fa-
lock"></i></span>
                                <input type="password" class="form-control"
id="password" name="password"
                                       required minlength="6">
                            </div>
                            <div class="form-text">Password must be at least 6
characters long.</div>
                        </div>

                        <div class="col-md-6 mb-3">
                            <label for="confirmPassword" class="form-
label">Confirm Password</label>
                            <div class="input-group">
                                <span class="input-group-text"><i class="fas fa-
lock"></i></span>
                                <input type="password" class="form-control"
id="confirmPassword"
                                       name="confirmPassword" required>
                            </div>
                        </div>
                    </div>

                    <div class="d-grid gap-2">
                        <button type="submit" class="btn btn-primary">
                            <i class="fas fa-user-plus"></i> Register
                        </button>
                    </div>
                </form>
```

```
            </div>
            <div class="card-footer text-center">
                <p class="mb-0">Already have an account? <a
href="${pageContext.request.contextPath}/login">Login</a></p>
            </div>
        </div>
    </div>
</div>

<script>
    document.getElementById('registerForm').addEventListener('submit',
function(event) {
        var password = document.getElementById('password').value;
        var confirmPassword = document.getElementById('confirmPassword').value;

        if (password !== confirmPassword) {
            event.preventDefault();
            alert('Passwords do not match!');
        }
    });
</script>

<!-- Include Footer -->
<jsp:include page="/WEB-INF/views/common/footer.jsp" />
```

3. Task Views

list.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="tm" uri="http://taskmanager.com/tags" %>

<!-- Include Header -->
<jsp:include page="/WEB-INF/views/common/header.jsp">
    <jsp:param name="title" value="Tasks" />
    <jsp:param name="activeNav" value="tasks" />
</jsp:include>

<div class="d-flex justify-content-between align-items-center mb-4">
    <h1><i class="fas fa-tasks"></i> Tasks</h1>
    <a href="${pageContext.request.contextPath}/tasks/add" class="btn btn-
primary">
        <i class="fas fa-plus"></i> Add Task
    </a>
</div>

<!-- Filter Form -->
<div class="card mb-4">
    <div class="card-header bg-light">
```

```
            <i class="fas fa-filter"></i> Filters
        </div>
        <div class="card-body">
            <form action="${pageContext.request.contextPath}/tasks" method="get"
id="filterForm">
                <div class="row">
                    <div class="col-md-3 mb-3">
                        <label for="status" class="form-label">Status</label>
                        <select class="form-select" id="status" name="status">
                            <option value="" ${empty selectedStatus ? 'selected' :
''}>All Statuses</option>
                            <option value="TODO" ${selectedStatus == 'TODO' ?
'selected' : ''}>To Do</option>
                            <option value="IN_PROGRESS" ${selectedStatus ==
'IN_PROGRESS' ? 'selected' : ''}>In Progress</option>
                            <option value="DONE" ${selectedStatus == 'DONE' ?
'selected' : ''}>Done</option>
                        </select>
                    </div>

                    <div class="col-md-3 mb-3">
                        <label for="priority" class="form-label">Priority</label>
                        <select class="form-select" id="priority" name="priority">
                            <option value="" ${empty selectedPriority ? 'selected' :
''}>All Priorities</option>
                            <option value="LOW" ${selectedPriority == 'LOW' ?
'selected' : ''}>Low</option>
                            <option value="MEDIUM" ${selectedPriority == 'MEDIUM' ?
'selected' : ''}>Medium</option>
                            <option value="HIGH" ${selectedPriority == 'HIGH' ?
'selected' : ''}>High</option>
                        </select>
                    </div>

                    <div class="col-md-3 mb-3">
                        <label for="categoryId" class="form-label">Category</label>
                        <select class="form-select" id="categoryId" name="categoryId">
                            <option value="" ${empty selectedCategoryId ? 'selected' :
''}>All Categories</option>
                            <c:forEach var="category" items="${categories}">
                                <option value="${category.id}"
                                        ${selectedCategoryId == category.id.toString()
? 'selected' : ''}>
                                    ${category.name}
                                </option>
                            </c:forEach>
                        </select>
                    </div>

                    <div class="col-md-3 mb-3">
                        <label for="search" class="form-label">Search</label>
                        <input type="text" class="form-control" id="search"
name="search"
                               placeholder="Search..." value="${searchTerm}">
```

```
                </div>
            </div>

            <div class="text-end">
                <button type="submit" class="btn btn-primary">
                    <i class="fas fa-search"></i> Apply Filters
                </button>
                <a href="${pageContext.request.contextPath}/tasks" class="btn btn-
secondary">
                    <i class="fas fa-undo"></i> Reset
                </a>
            </div>
        </form>
    </div>
</div>

<!-- Tasks Table -->
<div class="card shadow">
    <div class="card-body">
        <c:choose>
            <c:when test="${not empty tasks}">
                <div class="table-responsive">
                    <table class="table table-striped table-hover">
                        <thead>
                            <tr>
                                <th>Title</th>
                                <th>Status</th>
                                <th>Priority</th>
                                <th>Category</th>
                                <th>Due Date</th>
                                <th>Assigned To</th>
                                <th>Actions</th>
                            </tr>
                        </thead>
                        <tbody>
                            <c:forEach var="task" items="${tasks}">
                                <tr>
                                    <td>
                                        <a
href="${pageContext.request.contextPath}/tasks/detail/${task.id}">
                                            ${task.title}
                                        </a>
                                    </td>
                                    <td>
                                        <span class="badge
bg-${task.statusClass}">
                                            ${task.status.replace('_', ' ')}
                                        </span>
                                    </td>
                                    <td>
                                        <span class="badge
bg-${task.priorityClass}">
                                            ${task.priority}
                                        </span>
```

```
                                    </td>
                                    <td>
                                        <c:if test="${not empty task.category}">
                                            <span class="badge" style="background-
color: ${task.category.color}">
                                                ${task.category.name}
                                            </span>
                                        </c:if>
                                    </td>
                                    <td>
                                        <c:if test="${task.dueDate != null}">
                                            <span class="${task.overdue ? 'text-
danger fw-bold' : ''}">
                                                <tm:formatDate
date="${task.dueDate}" relativeFormat="true" />
                                            </span>
                                        </c:if>
                                    </td>
                                    <td>
                                        <c:if test="${not empty task.assignee}">
                                            ${task.assignee.fullName}
                                        </c:if>
                                    </td>
                                    <td>
                                        <div class="btn-group btn-group-sm">
                                            <a
href="${pageContext.request.contextPath}/tasks/edit/${task.id}"
                                               class="btn btn-outline-primary"
title="Edit">
                                                <i class="fas fa-edit"></i>
                                            </a>
                                            <button type="button" class="btn btn-
outline-danger delete-task"
                                                    data-id="${task.id}" data-
title="${task.title}" title="Delete">
                                                <i class="fas fa-trash"></i>
                                            </button>
                                        </div>
                                    </td>
                                </tr>
                            </c:forEach>
                        </tbody>
                    </table>
                </div>
            </c:when>
            <c:otherwise>
                <div class="alert alert-info mb-0">
                    <i class="fas fa-info-circle"></i> No tasks found matching
your criteria.
                </div>
            </c:otherwise>
        </c:choose>
    </div>
</div>
```

```
<!-- Delete Confirmation Modal -->
<div class="modal fade" id="deleteTaskModal" tabindex="-1">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header bg-danger text-white">
                <h5 class="modal-title"><i class="fas fa-exclamation-triangle">
</i> Confirm Delete</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <p>Are you sure you want to delete the task: <strong
id="taskTitle"></strong>?</p>
                <p class="mb-0 text-danger">This action cannot be undone!</p>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Cancel</button>
                <form id="deleteTaskForm" method="post">
                    <input type="hidden" name="action" value="delete">
                    <button type="submit" class="btn btn-danger">Delete</button>
                </form>
            </div>
        </div>
    </div>
</div>

<script>
    // Task deletion
    document.addEventListener('DOMContentLoaded', function() {
        const deleteModal = new
bootstrap.Modal(document.getElementById('deleteTaskModal'));

        document.querySelectorAll('.delete-task').forEach(button => {
            button.addEventListener('click', function() {
                const taskId = this.getAttribute('data-id');
                const taskTitle = this.getAttribute('data-title');

                document.getElementById('taskTitle').textContent = taskTitle;
                document.getElementById('deleteTaskForm').action =
                    '${pageContext.request.contextPath}/tasks/delete/' + taskId;

                deleteModal.show();
            });
        });
    });
</script>

<!-- Include Footer -->
<jsp:include page="/WEB-INF/views/common/footer.jsp" />
```

form.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<!-- Include Header -->
<jsp:include page="/WEB-INF/views/common/header.jsp">
    <jsp:param name="title" value="${formTitle}" />
    <jsp:param name="activeNav" value="tasks" />
</jsp:include>

<div class="d-flex justify-content-between align-items-center mb-4">
    <h1><i class="fas fa-${submitAction == 'create' ? 'plus-circle' : 'edit'}">
</i> ${formTitle}</h1>
    <a href="${pageContext.request.contextPath}/tasks" class="btn btn-secondary">
        <i class="fas fa-arrow-left"></i> Back to Tasks
    </a>
</div>

<div class="card shadow">
    <div class="card-body">
        <form action="${pageContext.request.contextPath}/tasks/${submitAction ==
'create' ? 'add' : 'edit'}"
              method="post" id="taskForm">

            <input type="hidden" name="action" value="${submitAction}">
            <input type="hidden" name="id" value="${task.id}">

            <div class="row">
                <div class="col-md-8 mb-3">
                    <label for="title" class="form-label">Title <span class="text-
danger">*</span></label>
                    <input type="text" class="form-control" id="title"
name="title"
                           value="${task.title}" required maxlength="100">
                </div>

                <div class="col-md-4 mb-3">
                    <label for="status" class="form-label">Status <span
class="text-danger">*</span></label>
                    <select class="form-select" id="status" name="status"
required>
                        <option value="TODO" ${task.status == 'TODO' ? 'selected'
: ''}>To Do</option>
                        <option value="IN_PROGRESS" ${task.status == 'IN_PROGRESS'
? 'selected' : ''}>In Progress</option>
                        <option value="DONE" ${task.status == 'DONE' ? 'selected'
: ''}>Done</option>
                    </select>
                </div>
            </div>

            <div class="row">
```

```
                <div class="col-md-4 mb-3">
                    <label for="priority" class="form-label">Priority <span
class="text-danger">*</span></label>
                    <select class="form-select" id="priority" name="priority"
required>
                        <option value="LOW" ${task.priority == 'LOW' ? 'selected'
: ''}>Low</option>
                        <option value="MEDIUM" ${task.priority == 'MEDIUM' ?
'selected' : ''}>Medium</option>
                        <option value="HIGH" ${task.priority == 'HIGH' ?
'selected' : ''}>High</option>
                    </select>
                </div>

                <div class="col-md-4 mb-3">
                    <label for="categoryId" class="form-label">Category</label>
                    <select class="form-select" id="categoryId" name="categoryId">
                        <option value="">-- Select Category --</option>
                        <c:forEach var="category" items="${categories}">
                            <option value="${category.id}"
                                    ${task.categoryId == category.id ? 'selected'
: ''}
                                    style="background-color: ${category.color}20">
                                ${category.name}
                            </option>
                        </c:forEach>
                    </select>
                </div>

                <div class="col-md-4 mb-3">
                    <label for="dueDate" class="form-label">Due Date</label>
                    <input type="date" class="form-control" id="dueDate"
name="dueDate"
                           value="<fmt:formatDate value='${task.dueDate}'
pattern='yyyy-MM-dd' />">
                </div>
            </div>

            <div class="mb-3">
                <label for="assigneeId" class="form-label">Assign To</label>
                <select class="form-select" id="assigneeId" name="assigneeId">
                    <option value="">-- Unassigned --</option>
                    <c:forEach var="user" items="${users}">
                        <option value="${user.id}" ${task.assigneeId == user.id ?
'selected' : ''}>
                            ${user.fullName} (${user.username})
                        </option>
                    </c:forEach>
                </select>
            </div>

            <div class="mb-3">
                <label for="description" class="form-label">Description</label>
                <textarea class="form-control" id="description" name="description"
```

```
                            rows="5">${task.description}</textarea>
            </div>

            <div class="text-end">
                <a href="${pageContext.request.contextPath}/tasks" class="btn btn-
secondary">
                    <i class="fas fa-times"></i> Cancel
                </a>
                <button type="submit" class="btn btn-primary">
                    <i class="fas fa-save"></i> Save Task
                </button>
            </div>
        </form>
    </div>
</div>

<!-- Include Footer -->
<jsp:include page="/WEB-INF/views/common/footer.jsp" />
```

detail.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="tm" uri="http://taskmanager.com/tags" %>

<!-- Include Header -->
<jsp:include page="/WEB-INF/views/common/header.jsp">
    <jsp:param name="title" value="Task Details" />
    <jsp:param name="activeNav" value="tasks" />
</jsp:include>

<div class="d-flex justify-content-between align-items-center mb-4">
    <h1><i class="fas fa-clipboard-list"></i> Task Details</h1>
    <div>
        <a href="${pageContext.request.contextPath}/tasks/edit/${task.id}"
class="btn btn-primary">
            <i class="fas fa-edit"></i> Edit
        </a>
        <a href="${pageContext.request.contextPath}/tasks" class="btn btn-
secondary">
            <i class="fas fa-arrow-left"></i> Back to Tasks
        </a>
    </div>
</div>

<div class="row">
    <div class="col-md-8">
        <!-- Task Information Card -->
        <div class="card shadow mb-4">
            <div class="card-header bg-light d-flex justify-content-between align-
items-center">
```

```
                <h5 class="mb-0"><i class="fas fa-info-circle"></i> Task
Information</h5>
                <div>
                    <span class="badge bg-${task.statusClass} me-
1">${task.status.replace('_', ' ')}</span>
                    <span class="badge bg-${task.priorityClass}">${task.priority}
</span>
                    <c:if test="${not empty task.category}">
                        <span class="badge ms-1" style="background-color:
${task.category.color}">
                            ${task.category.name}
                        </span>
                    </c:if>
                </div>
            </div>
            <div class="card-body">
                <h3 class="mb-3">${task.title}</h3>

                <div class="row mb-3">
                    <div class="col-md-4">
                        <strong><i class="fas fa-calendar"></i> Created:</strong>
                        <tm:formatDate date="${task.createdAt}" pattern="MMM d,
yyyy h:mm a" />
                    </div>
                    <div class="col-md-4">
                        <strong><i class="fas fa-clock"></i> Updated:</strong>
                        <tm:formatDate date="${task.updatedAt}" pattern="MMM d,
yyyy h:mm a" />
                    </div>
                    <div class="col-md-4">
                        <strong><i class="fas fa-calendar-alt"></i> Due Date:
</strong>
                        <c:choose>
                            <c:when test="${task.dueDate != null}">
                                <span class="${task.overdue ? 'text-danger fw-
bold' : ''}">
                                    <tm:formatDate date="${task.dueDate}"
pattern="MMM d, yyyy" />
                                    <c:if test="${task.overdue}">
                                        <i class="fas fa-exclamation-triangle
text-danger ms-1"
                                           title="Overdue"></i>
                                    </c:if>
                                </span>
                            </c:when>
                            <c:otherwise>
                                <span class="text-muted">Not set</span>
                            </c:otherwise>
                        </c:choose>
                    </div>
                </div>

                <div class="mb-4">
                    <h5><i class="fas fa-align-left"></i> Description</h5>
```

```
                    <div class="p-3 bg-light rounded">
                        <c:choose>
                            <c:when test="${not empty task.description}">
                                <p class="mb-0 whitespace-pre-
wrap">${task.description}</p>
                            </c:when>
                            <c:otherwise>
                                <p class="text-muted mb-0">No description
provided.</p>
                            </c:otherwise>
                        </c:choose>
                    </div>
                </div>

                <!-- Task Comments -->
                <div class="mt-4">
                    <h5><i class="fas fa-comments"></i> Comments</h5>

                    <c:choose>
                        <c:when test="${not empty comments}">
                            <div class="comment-list">
                                <c:forEach var="comment" items="${comments}">
                                    <div class="comment card mb-2">
                                        <div class="card-body py-2">
                                            <div class="d-flex justify-content-
between align-items-center mb-2">
                                                <div>

<strong>${comment.user.fullName}</strong>
                                                    <small class="text-muted ms-
2">
                                                        <tm:formatDate
date="${comment.createdAt}" relativeFormat="true" />
                                                    </small>
                                                </div>
                                                <c:if test="${comment.user.id ==
sessionScope.user.id || sessionScope.user.role == 'ADMIN'}">
                                                    <button class="btn btn-sm btn-
outline-danger delete-comment"
                                                            data-
id="${comment.id}" title="Delete comment">
                                                        <i class="fas fa-times">
</i>
                                                    </button>
                                                </c:if>
                                            </div>
                                            <p class="mb-0">${comment.comment}</p>
                                        </div>
                                    </div>
                                </c:forEach>
                            </div>
                        </c:when>
                        <c:otherwise>
                            <p class="text-muted">No comments yet.</p>
```

```
                        </c:otherwise>
                    </c:choose>

                    <!-- Add Comment Form -->
                    <form
action="${pageContext.request.contextPath}/tasks/comment/${task.id}" method="post"
class="mt-3">
                        <div class="mb-3">
                            <label for="comment" class="form-label">Add a
comment</label>
                            <textarea class="form-control" id="comment"
name="comment" rows="3" required></textarea>
                        </div>
                        <div class="text-end">
                            <button type="submit" class="btn btn-primary">
                                <i class="fas fa-paper-plane"></i> Post Comment
                            </button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>

    <div class="col-md-4">
        <!-- Task Metadata Card -->
        <div class="card shadow mb-4">
            <div class="card-header bg-light">
                <h5 class="mb-0"><i class="fas fa-user-tag"></i> Assigned
People</h5>
            </div>
            <div class="card-body">
                <div class="mb-3">
                    <strong><i class="fas fa-user-edit"></i> Created by:</strong>
                    <div class="d-flex align-items-center mt-1">
                        <span class="avatar avatar-sm bg-primary me-2">
                            ${fn:substring(task.creator.fullName, 0,
1).toUpperCase()}
                        </span>
                        <span>${task.creator.fullName}</span>
                    </div>
                </div>

                <div>
                    <strong><i class="fas fa-user-check"></i> Assigned to:
</strong>
                    <div class="d-flex align-items-center mt-1">
                        <c:choose>
                            <c:when test="${not empty task.assignee}">
                                <span class="avatar avatar-sm bg-info me-2">
                                    ${fn:substring(task.assignee.fullName, 0,
1).toUpperCase()}
                                </span>
                                <span>${task.assignee.fullName}</span>
```

```
                            </c:when>
                            <c:otherwise>
                                <span class="text-muted">Unassigned</span>
                            </c:otherwise>
                        </c:choose>
                    </div>
                </div>
            </div>
        </div>

        <!-- Task Attachments Card -->
        <div class="card shadow mb-4">
            <div class="card-header bg-light">
                <h5 class="mb-0"><i class="fas fa-paperclip"></i> Attachments</h5>
            </div>
            <div class="card-body">
                <c:choose>
                    <c:when test="${not empty attachments}">
                        <ul class="list-group">
                            <c:forEach var="attachment" items="${attachments}">
                                <li class="list-group-item d-flex justify-content-
between align-items-center">
                                    <div>
                                        <i class="fas
${attachment.fileType.startsWith('image') ? 'fa-image' : 'fa-file'}"></i>
                                        <a
href="${pageContext.request.contextPath}/tasks/attachment/${attachment.id}"
                                           target="_blank" class="ms-2">
                                            ${attachment.fileName}
                                        </a>
                                        <small class="text-muted d-block">
                                            ${attachment.fileSizeFormatted} •
Uploaded by ${attachment.uploader.fullName}
                                        </small>
                                    </div>
                                    <button class="btn btn-sm btn-outline-danger
delete-attachment"
                                            data-id="${attachment.id}"
title="Delete attachment">
                                        <i class="fas fa-trash"></i>
                                    </button>
                                </li>
                            </c:forEach>
                        </ul>
                    </c:when>
                    <c:otherwise>
                        <p class="text-muted">No attachments yet.</p>
                    </c:otherwise>
                </c:choose>

                <!-- Add Attachment Form -->
                <form
action="${pageContext.request.contextPath}/tasks/attachment/${task.id}"
                      method="post" enctype="multipart/form-data" class="mt-3">
```

```
                    <div class="mb-3">
                        <label for="file" class="form-label">Add
attachment</label>
                        <input type="file" class="form-control" id="file"
name="file" required>
                    </div>
                    <div class="text-end">
                        <button type="submit" class="btn btn-primary">
                            <i class="fas fa-upload"></i> Upload
                        </button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>

<!-- Delete Comment Confirmation Modal -->
<div class="modal fade" id="deleteCommentModal" tabindex="-1">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header bg-danger text-white">
                <h5 class="modal-title">Delete Comment</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <p>Are you sure you want to delete this comment?</p>
                <p class="mb-0 text-danger">This action cannot be undone!</p>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Cancel</button>
                <form id="deleteCommentForm" method="post">
                    <input type="hidden" name="action" value="deleteComment">
                    <button type="submit" class="btn btn-danger">Delete</button>
                </form>
            </div>
        </div>
    </div>
</div>

<!-- Delete Attachment Confirmation Modal -->
<div class="modal fade" id="deleteAttachmentModal" tabindex="-1">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header bg-danger text-white">
                <h5 class="modal-title">Delete Attachment</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <p>Are you sure you want to delete this attachment?</p>
                <p class="mb-0 text-danger">This action cannot be undone!</p>
```

```
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Cancel</button>
                <form id="deleteAttachmentForm" method="post">
                    <input type="hidden" name="action" value="deleteAttachment">
                    <button type="submit" class="btn btn-danger">Delete</button>
                </form>
            </div>
        </div>
    </div>
</div>

<script>
    // Comment deletion
    document.addEventListener('DOMContentLoaded', function() {
        const commentModal = new
bootstrap.Modal(document.getElementById('deleteCommentModal'));
        const attachmentModal = new
bootstrap.Modal(document.getElementById('deleteAttachmentModal'));

        // Set up comment deletion
        document.querySelectorAll('.delete-comment').forEach(button => {
            button.addEventListener('click', function() {
                const commentId = this.getAttribute('data-id');
                document.getElementById('deleteCommentForm').action =
                    '${pageContext.request.contextPath}/tasks/comment/delete/' +
commentId + '?taskId=${task.id}';
                commentModal.show();
            });
        });

        // Set up attachment deletion
        document.querySelectorAll('.delete-attachment').forEach(button => {
            button.addEventListener('click', function() {
                const attachmentId = this.getAttribute('data-id');
                document.getElementById('deleteAttachmentForm').action =
                    '${pageContext.request.contextPath}/tasks/attachment/delete/'
+ attachmentId + '?taskId=${task.id}';
                attachmentModal.show();
            });
        });
    });
</script>

<!-- Include Footer -->
<jsp:include page="/WEB-INF/views/common/footer.jsp" />
```

6.6 API Implementation

Let's implement the REST API for our task management system.

TaskApiServlet.java:

```
package com.taskmanager.web.servlet.api;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.taskmanager.exception.ServiceException;
import com.taskmanager.exception.ValidationException;
import com.taskmanager.model.Task;
import com.taskmanager.service.TaskService;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

@WebServlet("/api/tasks/*")
public class TaskApiServlet extends HttpServlet {

    private static final Logger logger =
LogManager.getLogger(TaskApiServlet.class);

    private TaskService taskService;
    private ObjectMapper objectMapper;

    @Override
    public void init() throws ServletException {
        taskService = new TaskService();
        objectMapper = new ObjectMapper();
        objectMapper.setDateFormat(new SimpleDateFormat("yyyy-MM-dd"));
    }

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

        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        try {
            String pathInfo = request.getPathInfo();

            if (pathInfo == null || pathInfo.equals("/")) {
                // Get task list with filters
                String status = request.getParameter("status");
                String categoryId = request.getParameter("categoryId");
                String priority = request.getParameter("priority");
```

```
                String searchTerm = request.getParameter("search");

                List<Task> tasks = taskService.findAllWithFilters(status,
categoryId, priority, searchTerm);
                objectMapper.writeValue(response.getWriter(), tasks);
            } else {
                // Get task by ID
                String id = pathInfo.substring(1);
                Task task = taskService.findById(Long.parseLong(id));

                if (task == null) {
                    sendError(response, HttpServletResponse.SC_NOT_FOUND, "Task
not found");
                } else {
                    objectMapper.writeValue(response.getWriter(), task);
                }
            }
        } catch (NumberFormatException e) {
            sendError(response, HttpServletResponse.SC_BAD_REQUEST, "Invalid task
ID format");
        } catch (ServiceException e) {
            logger.error("Error retrieving tasks", e);
            sendError(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Error retrieving tasks: " + e.getMessage());
        }
    }

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

        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        try {
            // Parse task from request body
            Task task = objectMapper.readValue(request.getInputStream(),
Task.class);

            // Create task
            Long taskId = taskService.createTask(task);

            // Get created task
            Task createdTask = taskService.findById(taskId);

            // Return created task
            response.setStatus(HttpServletResponse.SC_CREATED);
            objectMapper.writeValue(response.getWriter(), createdTask);

        } catch (ValidationException e) {
            sendError(response, HttpServletResponse.SC_BAD_REQUEST,
e.getMessage());
        } catch (ServiceException e) {
```

```
            logger.error("Error creating task", e);
            sendError(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Error creating task: " + e.getMessage());
        } catch (Exception e) {
            logger.error("Unexpected error", e);
            sendError(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Unexpected error: " + e.getMessage());
        }
    }

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

        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        try {
            // Get task ID from path
            String pathInfo = request.getPathInfo();

            if (pathInfo == null || pathInfo.equals("/")) {
                sendError(response, HttpServletResponse.SC_BAD_REQUEST, "Task ID
is required");
                return;
            }

            // Parse ID
            Long taskId = Long.parseLong(pathInfo.substring(1));

            // Check if task exists
            Task existingTask = taskService.findById(taskId);

            if (existingTask == null) {
                sendError(response, HttpServletResponse.SC_NOT_FOUND, "Task not
found");
                return;
            }

            // Parse task from request body
            Task task = objectMapper.readValue(request.getInputStream(),
Task.class);
            task.setId(taskId);

            // Update task
            taskService.updateTask(task);

            // Get updated task
            Task updatedTask = taskService.findById(taskId);

            // Return updated task
            objectMapper.writeValue(response.getWriter(), updatedTask);

        } catch (NumberFormatException e) {
```

```
            sendError(response, HttpServletResponse.SC_BAD_REQUEST, "Invalid task
ID format");
        } catch (ValidationException e) {
            sendError(response, HttpServletResponse.SC_BAD_REQUEST,
e.getMessage());
        } catch (ServiceException e) {
            logger.error("Error updating task", e);
            sendError(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Error updating task: " + e.getMessage());
        } catch (Exception e) {
            logger.error("Unexpected error", e);
            sendError(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Unexpected error: " + e.getMessage());
        }
    }

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

        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");

        try {
            // Get task ID from path
            String pathInfo = request.getPathInfo();

            if (pathInfo == null || pathInfo.equals("/")) {
                sendError(response, HttpServletResponse.SC_BAD_REQUEST, "Task ID
is required");
                return;
            }

            // Parse ID
            Long taskId = Long.parseLong(pathInfo.substring(1));

            // Check if task exists
            Task existingTask = taskService.findById(taskId);

            if (existingTask == null) {
                sendError(response, HttpServletResponse.SC_NOT_FOUND, "Task not
found");
                return;
            }

            // Delete task
            taskService.deleteTask(taskId);

            // Return success response
            response.setStatus(HttpServletResponse.SC_NO_CONTENT);

        } catch (NumberFormatException e) {
            sendError(response, HttpServletResponse.SC_BAD_REQUEST, "Invalid task
```

```
ID format");
        } catch (ServiceException e) {
            logger.error("Error deleting task", e);
            sendError(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Error deleting task: " + e.getMessage());
        } catch (Exception e) {
            logger.error("Unexpected error", e);
            sendError(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Unexpected error: " + e.getMessage());
        }
    }

    private void sendError(HttpServletResponse response, int status, String
message) throws IOException {
        response.setStatus(status);
        Map<String, String> error = new HashMap<>();
        error.put("error", message);
        objectMapper.writeValue(response.getWriter(), error);
    }
}
```

Similar API servlets should be implemented for categories, users, comments, and attachments.

6.7 Front-End Integration

Let's integrate our API with JavaScript for a more interactive user experience.

script.js:

```
/**
 * Task Manager Application JavaScript
 */
document.addEventListener('DOMContentLoaded', function() {
    // Initialize tooltips
    var tooltipTriggerList = [].slice.call(document.querySelectorAll('[data-bs-
toggle="tooltip"]'));
    tooltipTriggerList.map(function(tooltipTriggerEl) {
        return new bootstrap.Tooltip(tooltipTriggerEl);
    });

    // Initialize form validation
    const forms = document.querySelectorAll('.needs-validation');

    Array.from(forms).forEach(form => {
        form.addEventListener('submit', event => {
            if (!form.checkValidity()) {
                event.preventDefault();
                event.stopPropagation();
            }

            form.classList.add('was-validated');
        }, false);
```

```
    });

    // Task status update via AJAX
    const statusButtons = document.querySelectorAll('.task-status-update');

    statusButtons.forEach(button => {
        button.addEventListener('click', function() {
            const taskId = this.getAttribute('data-task-id');
            const newStatus = this.getAttribute('data-status');

            updateTaskStatus(taskId, newStatus);
        });
    });

    // Task priority update via AJAX
    const prioritySelects = document.querySelectorAll('.task-priority-update');

    prioritySelects.forEach(select => {
        select.addEventListener('change', function() {
            const taskId = this.getAttribute('data-task-id');
            const newPriority = this.value;

            updateTaskPriority(taskId, newPriority);
        });
    });

    // Task assignment via AJAX
    const assigneeSelects = document.querySelectorAll('.task-assignee-update');

    assigneeSelects.forEach(select => {
        select.addEventListener('change', function() {
            const taskId = this.getAttribute('data-task-id');
            const newAssigneeId = this.value;

            updateTaskAssignee(taskId, newAssigneeId);
        });
    });

    // Auto-dismiss alerts after 5 seconds
    const alerts = document.querySelectorAll('.alert-auto-dismiss');

    alerts.forEach(alert => {
        setTimeout(() => {
            const bsAlert = new bootstrap.Alert(alert);
            bsAlert.close();
        }, 5000);
    });
});

/**
 * Update task status via AJAX
 */
function updateTaskStatus(taskId, status) {
    fetch(`/task-manager/api/tasks/${taskId}`, {
```

```
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            id: taskId,
            status: status
        })
    })
    .then(response => {
        if (!response.ok) {
            return response.json().then(data => {
                throw new Error(data.error || 'Failed to update task status');
            });
        }
        return response.json();
    })
    .then(data => {
        // Update UI
        const statusBadge = document.querySelector(`.task-status-badge[data-task-
id="${taskId}"]`);
        if (statusBadge) {
            statusBadge.textContent = status.replace('_', ' ');
            statusBadge.className = `badge task-status-badge
bg-${getStatusClass(status)}`;
        }

        // Show success message
        showToast('Success', 'Task status updated successfully', 'success');
    })
    .catch(error => {
        console.error('Error updating task status:', error);
        showToast('Error', error.message, 'danger');
    });
}

/**
 * Update task priority via AJAX
 */
function updateTaskPriority(taskId, priority) {
    fetch(`/task-manager/api/tasks/${taskId}`, {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            id: taskId,
            priority: priority
        })
    })
    .then(response => {
        if (!response.ok) {
            return response.json().then(data => {
                throw new Error(data.error || 'Failed to update task priority');
```

```
            });
        }
        return response.json();
    })
    .then(data => {
        // Update UI
        const priorityBadge = document.querySelector(`.task-priority-badge[data-
task-id="${taskId}"]`);
        if (priorityBadge) {
            priorityBadge.textContent = priority;
            priorityBadge.className = `badge task-priority-badge
bg-${getPriorityClass(priority)}`;
        }

        // Show success message
        showToast('Success', 'Task priority updated successfully', 'success');
    })
    .catch(error => {
        console.error('Error updating task priority:', error);
        showToast('Error', error.message, 'danger');
    });
}

/**
 * Update task assignee via AJAX
 */
function updateTaskAssignee(taskId, assigneeId) {
    fetch(`/task-manager/api/tasks/${taskId}`, {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            id: taskId,
            assigneeId: assigneeId || null
        })
    })
    .then(response => {
        if (!response.ok) {
            return response.json().then(data => {
                throw new Error(data.error || 'Failed to update task assignee');
            });
        }
        return response.json();
    })
    .then(data => {
        // Show success message
        showToast('Success', 'Task assignment updated successfully', 'success');
    })
    .catch(error => {
        console.error('Error updating task assignee:', error);
        showToast('Error', error.message, 'danger');
    });
}
```

```
/**
 * Show toast notification
 */
function showToast(title, message, type = 'info') {
    const toastContainer = document.getElementById('toast-container');

    if (!toastContainer) {
        // Create toast container if it doesn't exist
        const container = document.createElement('div');
        container.id = 'toast-container';
        container.className = 'toast-container position-fixed bottom-0 end-0 p-3';
        document.body.appendChild(container);
    }

    const toastId = 'toast-' + Date.now();
    const toastHtml = `
        <div id="${toastId}" class="toast" role="alert" aria-live="assertive"
aria-atomic="true">
            <div class="toast-header bg-${type} text-white">
                <strong class="me-auto">${title}</strong>
                <button type="button" class="btn-close btn-close-white" data-bs-
dismiss="toast" aria-label="Close"></button>
            </div>
            <div class="toast-body">
                ${message}
            </div>
        </div>
    `;

    document.getElementById('toast-container').insertAdjacentHTML('beforeend',
toastHtml);

    const toastElement = document.getElementById(toastId);
    const toast = new bootstrap.Toast(toastElement, { autohide: true, delay: 5000
});
    toast.show();

    // Remove toast element after it's hidden
    toastElement.addEventListener('hidden.bs.toast', function() {
        toastElement.remove();
    });
}

/**
 * Get Bootstrap class for task status
 */
function getStatusClass(status) {
    switch (status) {
        case 'TODO': return 'secondary';
        case 'IN_PROGRESS': return 'primary';
        case 'DONE': return 'success';
        default: return 'light';
    }
```

```
}

/**
 * Get Bootstrap class for task priority
 */
function getPriorityClass(priority) {
    switch (priority) {
        case 'HIGH': return 'danger';
        case 'MEDIUM': return 'warning';
        case 'LOW': return 'info';
        default: return 'secondary';
    }
}
```

6.8 Building and Deploying

Build and deploy process:

1. Compile and package the application:

```
mvn clean package
```

2. Deploy the WAR file to Tomcat:

```
mvn tomcat7:deploy
```

3. Or manually copy the WAR file to Tomcat's webapps directory:

```
cp target/task-manager.war $TOMCAT_HOME/webapps/
```

4. Access the application at `http://localhost:8080/task-manager/`

7. Reference Section

7.1 Important API Methods

Servlet API Core Methods

HttpServlet Methods:

Method	Description
`doGet(HttpServletRequest req, HttpServletResponse resp)`	Handles HTTP GET requests

Method	Description
`doPost(HttpServletRequest req, HttpServletResponse resp)`	Handles HTTP POST requests
`doPut(HttpServletRequest req, HttpServletResponse resp)`	Handles HTTP PUT requests
`doDelete(HttpServletRequest req, HttpServletResponse resp)`	Handles HTTP DELETE requests
`service(HttpServletRequest req, HttpServletResponse resp)`	Dispatches to appropriate do* method
`init(ServletConfig config)`	Called by container when servlet is initialized
`destroy()`	Called by container before servlet is taken out of service
`getServletConfig()`	Returns ServletConfig object
`getServletContext()`	Returns ServletContext object

HttpServletRequest Methods:

Method	Description
`getParameter(String name)`	Returns parameter value for the specified name
`getParameterNames()`	Returns names of all parameters
`getParameterValues(String name)`	Returns all values for the specified parameter
`getParameterMap()`	Returns map of all parameters
`getAttribute(String name)`	Returns attribute value for the specified name
`setAttribute(String name, Object value)`	Sets an attribute
`removeAttribute(String name)`	Removes an attribute
`getSession()`	Returns current session or creates a new one
`getSession(boolean create)`	Returns current session or optionally creates a new one
`getCookies()`	Returns all cookies
`getHeader(String name)`	Returns header value for the specified name
`getHeaderNames()`	Returns names of all headers
`getMethod()`	Returns HTTP method
`getPathInfo()`	Returns path info
`getQueryString()`	Returns query string

Method	Description
`getRequestURI()`	Returns request URI
`getRequestURL()`	Returns request URL
`getServletPath()`	Returns servlet path
`getContextPath()`	Returns context path
`getInputStream()`	Returns input stream for reading request body
`getReader()`	Returns reader for reading request body

HttpServletResponse Methods:

Method	Description
`setStatus(int sc)`	Sets status code
`sendError(int sc)`	Sends error status code
`sendError(int sc, String msg)`	Sends error status code with message
`sendRedirect(String location)`	Sends redirect response
`setContentType(String type)`	Sets content type
`setContentLength(int len)`	Sets content length
`setCharacterEncoding(String charset)`	Sets character encoding
`getWriter()`	Returns writer for writing response
`getOutputStream()`	Returns output stream for writing binary response
`addCookie(Cookie cookie)`	Adds cookie to response
`setHeader(String name, String value)`	Sets header
`addHeader(String name, String value)`	Adds header
`setDateHeader(String name, long date)`	Sets date header
`addDateHeader(String name, long date)`	Adds date header
`setIntHeader(String name, int value)`	Sets integer header
`addIntHeader(String name, int value)`	Adds integer header

ServletContext Methods:

Method	Description
`getAttribute(String name)`	Returns attribute value for the specified name
`getAttributeNames()`	Returns names of all attributes

Method	Description
`setAttribute(String name, Object object)`	Sets an attribute
`removeAttribute(String name)`	Removes an attribute
`getInitParameter(String name)`	Returns init parameter value for the specified name
`getInitParameterNames()`	Returns names of all init parameters
`getContext(String uripath)`	Returns ServletContext for the specified URI path
`getContextPath()`	Returns context path
`getMajorVersion()`	Returns major version of Servlet API
`getMinorVersion()`	Returns minor version of Servlet API
`getServerInfo()`	Returns server info
`getServletContextName()`	Returns servlet context name
`log(String msg)`	Writes message to log
`log(String message, Throwable throwable)`	Writes message and stack trace to log
`getRealPath(String path)`	Returns file system path for the specified path
`getResourceAsStream(String path)`	Returns input stream for the specified resource
`getResourcePaths(String path)`	Returns paths for resources under the specified path
`getResource(String path)`	Returns URL for the specified resource
`getRequestDispatcher(String path)`	Returns RequestDispatcher for the specified path
`getNamedDispatcher(String name)`	Returns RequestDispatcher for the specified servlet name
`getServletRegistration(String servletName)`	Returns ServletRegistration for the specified servlet name
`getServletRegistrations()`	Returns map of ServletRegistration objects
`getFilterRegistration(String filterName)`	Returns FilterRegistration for the specified filter name
`getFilterRegistrations()`	Returns map of FilterRegistration objects
`getSessionCookieConfig()`	Returns SessionCookieConfig object

Method	Description
`setSessionTrackingModes(Set<SessionTrackingMode> sessionTrackingModes)`	Sets session tracking modes
`getDefaultSessionTrackingModes()`	Returns default session tracking modes
`getEffectiveSessionTrackingModes()`	Returns effective session tracking modes

HttpSession Methods:

Method	Description
`getAttribute(String name)`	Returns attribute value for the specified name
`getAttributeNames()`	Returns names of all attributes
`setAttribute(String name, Object value)`	Sets an attribute
`removeAttribute(String name)`	Removes an attribute
`getId()`	Returns session ID
`getCreationTime()`	Returns creation time
`getLastAccessedTime()`	Returns last accessed time
`getMaxInactiveInterval()`	Returns max inactive interval
`setMaxInactiveInterval(int interval)`	Sets max inactive interval
`invalidate()`	Invalidates session
`isNew()`	Returns whether session is new

JSP Implicit Objects

Object	Type	Description
`request`	`javax.servlet.http.HttpServletRequest`	Request object
`response`	`javax.servlet.http.HttpServletResponse`	Response object
`out`	`javax.servlet.jsp.JspWriter`	Writer for output
`session`	`javax.servlet.http.HttpSession`	Session object
`application`	`javax.servlet.ServletContext`	ServletContext object
`config`	`javax.servlet.ServletConfig`	ServletConfig object
`pageContext`	`javax.servlet.jsp.PageContext`	PageContext object
`page`	`java.lang.Object` (this)	Page instance
`exception`	`java.lang.Throwable`	Exception (only in error pages)

JSTL Core Tags

Tag	Description
`<c:out>`	Outputs expression with optional escaping
`<c:set>`	Sets variable in scope
`<c:remove>`	Removes variable from scope
`<c:if>`	Conditional tag
`<c:choose>`, `<c:when>`, `<c:otherwise>`	Switch-like construct
`<c:forEach>`	Iteration tag
`<c:forTokens>`	Iterates over tokens
`<c:url>`	Creates URL with optional parameters
`<c:param>`	Adds parameter to URL
`<c:redirect>`	Redirects to URL
`<c:import>`	Imports content from URL
`<c:catch>`	Catches exceptions

7.2 Deployment Descriptor Configurations

web.xml Elements:

```
<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">

    <!-- Display Name -->
    <display-name>Task Manager</display-name>

    <!-- Context Parameters -->
    <context-param>
        <param-name>appName</param-name>
        <param-value>Task Manager</param-value>
    </context-param>

    <!-- Servlet Definition -->
    <servlet>
        <servlet-name>TaskListServlet</servlet-name>
        <servlet-class>com.taskmanager.web.servlet.task.TaskListServlet</servlet-
class>
        <init-param>
            <param-name>defaultPageSize</param-name>
            <param-value>10</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
```

```
    </servlet>

    <!-- Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>TaskListServlet</servlet-name>
        <url-pattern>/tasks</url-pattern>
    </servlet-mapping>

    <!-- Filter Definition -->
    <filter>
        <filter-name>AuthenticationFilter</filter-name>
        <filter-class>com.taskmanager.web.filter.AuthenticationFilter</filter-
class>
    </filter>

    <!-- Filter Mapping -->
    <filter-mapping>
        <filter-name>AuthenticationFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- Listener Definition -->
    <listener>
        <listener-class>com.taskmanager.web.listener.AppInitListener</listener-
class>
    </listener>

    <!-- Session Configuration -->
    <session-config>
        <session-timeout>30</session-timeout>
        <cookie-config>
            <http-only>true</http-only>
            <secure>true</secure>
        </cookie-config>
        <tracking-mode>COOKIE</tracking-mode>
    </session-config>

    <!-- Welcome Files -->
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <!-- Error Pages -->
    <error-page>
        <error-code>404</error-code>
        <location>/error/404.jsp</location>
    </error-page>

    <error-page>
        <error-code>500</error-code>
        <location>/error/500.jsp</location>
    </error-page>

    <error-page>
```

```
        <exception-type>java.lang.Exception</exception-type>
        <location>/error/exception.jsp</location>
    </error-page>

    <!-- Security Constraints -->
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Admin Area</web-resource-name>
            <url-pattern>/admin/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>admin</role-name>
        </auth-constraint>
    </security-constraint>

    <!-- Login Configuration -->
    <login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/login.jsp</form-login-page>
            <form-error-page>/login-error.jsp</form-error-page>
        </form-login-config>
    </login-config>

    <!-- Security Roles -->
    <security-role>
        <role-name>admin</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role>

    <!-- MIME Mappings -->
    <mime-mapping>
        <extension>pdf</extension>
        <mime-type>application/pdf</mime-type>
    </mime-mapping>

    <!-- JSP Config -->
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <page-encoding>UTF-8</page-encoding>
            <scripting-invalid>false</scripting-invalid>
            <include-prelude>/WEB-INF/jspf/header.jspf</include-prelude>
            <include-coda>/WEB-INF/jspf/footer.jspf</include-coda>
        </jsp-property-group>

        <taglib>
            <taglib-uri>http://taskmanager.com/tags</taglib-uri>
            <taglib-location>/WEB-INF/taglib/taskmanager.tld</taglib-location>
        </taglib>
```

```
    </jsp-config>

    <!-- Resource References -->
    <resource-ref>
        <res-ref-name>jdbc/TaskManagerDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</web-app>
```

Common Annotations:

Annotation	Description
`@WebServlet`	Declares a servlet
`@WebFilter`	Declares a filter
`@WebListener`	Declares a listener
`@WebInitParam`	Declares an initialization parameter
`@ServletSecurity`	Declares security constraints
`@MultipartConfig`	Configures multipart/form-data processing
`@HttpMethodConstraint`	Constrains HTTP methods
`@HttpConstraint`	Specifies security constraints
`@DeclareRoles`	Declares security roles

7.3 Troubleshooting Common Errors

1. HTTP Status Codes

Status Code	Description	Common Causes	Solutions
400 (Bad Request)	Client sent invalid request	Invalid form data, invalid JSON, missing required parameters	Validate input data, check request format
401 (Unauthorized)	Authentication required	Missing or invalid credentials	Implement proper authentication, check credentials
403 (Forbidden)	No permission to access resource	Insufficient privileges	Implement proper authorization, check user roles
404 (Not Found)	Resource not found	Invalid URL, deleted resource	Check URL paths, verify resource exists
405 (Method Not Allowed)	HTTP method not supported	Using wrong HTTP method	Use correct HTTP method for resource

Status Code	Description	Common Causes	Solutions
500 (Internal Server Error)	Server error	Exceptions in servlet code, database errors	Check logs, add error handling, fix bugs
503 (Service Unavailable)	Server temporarily unavailable	Server overload, maintenance	Scale resources, implement retry mechanisms

2. Common Exceptions

Exception	Description	Common Causes	Solutions
`ServletException`	General servlet exception	Servlet initialization failures, request processing errors	Check servlet configuration, add error handling
`IOException`	I/O error	Network issues, disk errors	Check connections, implement timeouts
`ClassNotFoundException`	Class not found	Missing libraries, incorrect classpath	Check dependencies, verify classpath
`NullPointerException`	Null object reference	Uninitialized variables, missing data	Add null checks, initialize variables
`SQLException`	Database error	Connection issues, query errors	Check database connection, validate SQL queries
`JspException`	JSP error	Tag errors, EL evaluation issues	Validate JSP syntax, check tag libraries
`IllegalArgumentException`	Invalid argument	Invalid parameters, data validation failures	Validate input data, add parameter checks
`UnsupportedEncodingException`	Encoding not supported	Invalid character encoding	Use standard encodings (UTF-8)
`AuthenticationException`	Authentication failed	Invalid credentials, expired session	Implement proper authentication, validate credentials

3. Debugging Techniques

1. **Enable Logging**:
    - Configure Log4j/Logback
    - Add log statements at key points
    - Set appropriate log levels (DEBUG, INFO, WARN, ERROR)
2. **Use Debug Mode in IDE**:
    - Set breakpoints

- Inspect variables
- Step through code

3. **Check Server Logs**:
   - Tomcat logs (catalina.out, localhost.log)
   - Application logs
4. **HTTP Request/Response Analysis**:
   - Use browser developer tools
   - Monitor network traffic
   - Examine request/response headers and bodies
5. **Database Query Analysis**:
   - Enable SQL logging
   - Check query execution plans
   - Monitor database connections

7.4 Performance Optimization Techniques

1. Servlet Optimization

Technique	Description	Implementation
Request Threading	Optimize thread handling	Use async processing, thread pooling
Response Caching	Cache responses	Use HTTP headers, implement caching filter
Response Compression	Compress response data	Enable GZIP compression in filter
Connection Pooling	Reuse connections	Use connection pool for databases, HTTP clients
Request Buffering	Buffer request/response	Use buffered streams, set buffer sizes
Asynchronous Processing	Handle requests asynchronously	Use `AsyncContext` for long-running operations
Lazy Loading	Load resources when needed	Implement lazy initialization patterns
Resource Reuse	Reuse expensive resources	Pool and reuse objects, connections

2. Database Optimization

Technique	Description	Implementation
Connection Pooling	Reuse database connections	Use HikariCP, DBCP, or container connection pools
Query Optimization	Optimize SQL queries	Use indexes, optimize joins, limit result sets

Technique	Description	Implementation
Statement Caching	Cache prepared statements	Enable statement caching in connection pool
Batch Processing	Process multiple operations	Use batch updates/inserts for multiple operations
Transaction Management	Optimize transactions	Use appropriate isolation levels, minimize transaction scope
Result Set Processing	Optimize result handling	Use scrollable/updatable result sets when needed
Database Sharding	Distribute data across servers	Implement sharding for large-scale applications
Read-Write Splitting	Separate read/write operations	Use read replicas for read operations

3. JSP Optimization

Technique	Description	Implementation
JSP Precompilation	Precompile JSPs	Compile JSPs during build
Output Buffering	Buffer JSP output	Configure buffer size
Template Reuse	Reuse common elements	Use includes, custom tags, template systems
Minimize Scriptlets	Reduce Java code in JSPs	Use JSTL, EL, custom tags
Error Page Optimization	Optimize error handling	Define appropriate error pages
JSP Caching	Cache JSP output	Implement output caching
Content Delivery	Optimize content delivery	Use CDNs for static content
Image Optimization	Optimize images	Compress images, use appropriate formats

4. Front-End Optimization

Technique	Description	Implementation
Minification	Reduce file size	Minify JS/CSS files
Bundling	Combine files	Bundle multiple JS/CSS files
Lazy Loading	Load resources when needed	Implement lazy loading for images, scripts
Caching	Cache resources	Set appropriate cache headers
Compression	Compress resources	Enable GZIP/Brotli compression
CDN Usage	Use content delivery networks	Distribute static content via CDNs
Asynchronous Loading	Load scripts asynchronously	Use async/defer attributes

Technique	Description	Implementation
Resource Hints	Optimize resource loading	Use preload, prefetch, preconnect hints

5. General Best Practices

1. **Implement Proper Logging**:
    - Use appropriate log levels
    - Avoid excessive logging
    - Implement log rotation
2. **Use Appropriate Data Structures**:
    - Choose efficient collections
    - Use memory-efficient objects
    - Implement caching where appropriate
3. **Optimize Session Management**:
    - Minimize session data
    - Implement session timeouts
    - Use session clustering for high availability
4. **Implement Monitoring**:
    - Monitor application metrics
    - Set up alerting
    - Analyze performance trends
5. **Regular Performance Testing**:
    - Load testing
    - Stress testing
    - Performance profiling
6. **Security Considerations**:
    - Implement secure coding practices
    - Protect against common vulnerabilities
    - Use security headers
7. **Code Optimization**:
    - Follow best practices
    - Refactor inefficient code
    - Use design patterns appropriately

This completes our comprehensive guide to Java EE Servlets and JSP. By understanding and applying these concepts, you can build robust, efficient, and maintainable web applications using Java EE technologies.