Servlet Session management

Servlet Session Management is a mechanism in Java used by Web container to store session information.

Session tracking is a way to manage the data of a user, this is known as session management in servlet.

Session in Java are managed through different ways, such as, HTTP Session API, Cookies, URL rewriting, etc.

Session management or tracking is an important feature of modern websites that allows server to remember clients.

HTTP and server are stateless. Every request- response is independent and new one.

But keeping track of client's activity across multiple requests is important in a session.

1. **Cookies**
2. **Hidden form field**
3. **URL Rewriting**
4. **HttpSession**

Session is used to store everything that we can get from the client from all the requests the client makes.



Using cookies for session management has several advantages.

* They are simple to implement and use, since they are supported by most browsers and web frameworks, and do not require any additional infrastructure or configuration.
* Cookies can reduce the server load and memory consumption, as well as enable load balancing and failover by being shared among different servers.
* Furthermore, cookies can persist the user's state and preferences across different sessions and devices, with the possibility of having a long expiration date or no expiration date at all.
* This allows the user to resume their session after closing and reopening the browser or after clearing the browser cache.

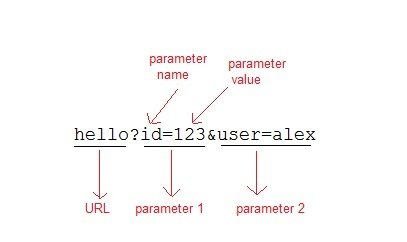
Cons of Cookies

* Using cookies for session management can have some drawbacks, such as security issues due to cookie theft, tampering.
* disabling or deleting cookies can prevent the web application from functioning properly.
* Additionally, some browsers may block or restrict third-party cookies which can affect the functionality of embedded features or services.

URL Rewriting:

If the client has disabled cookies in the browser, then session management using cookie wont work. In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work.

In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal(=) sign.



When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL. The **Web Container** will fetch the extra part of the requested URL and use it for session management.

The getParameter() method is used to get the parameter value at the server side.

Hidden Form Fields for Session Management

Hidden form field can also be used to store session information for a particular client.

In case of hidden form field, a hidden field is used to store client state.

In this case user information is stored in hidden field value and retrieved from another servlet.

Advantages of Using Hidden Form Field for Session Management

* Does not have to depend on browser whether the cookie is disabled or not.
* Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.

Disadvantage of Using Hidden Form Field for Session Management

* Extra form submission is required on every page. This is a big overhead.

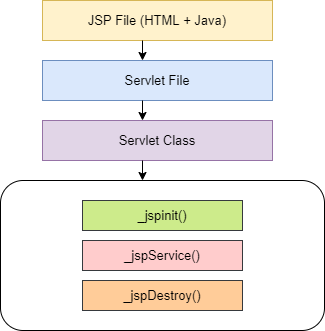
Session tracking in servlet is very simple and it involves following steps

* Get the associated session object (HttpSession) using *request.getSession()*.
* To get the specific value out of session object, call *getAttribute(String)* on the HttpSession object.
* To store any information in a session call setAttribute(key,object) on a session object.
* To remove the session data , call removeAttribute(key) to discard a object with a given key.
* To  invalidate  the session, call invalidate() on session object. This is used to  logout the logged in user.

HttpSession s=new HttpSession(true);

HttpSession s=new HttpSession(false);

HttpSession s=new HttpSession();



**JSP Directives** control the processing of an entire JSP page.

It gives directions to the server regarding processing of a page.

There are three types of directives available in JSP: page, include and taglib.

Three types of directives

* 1. Page directives
  2. Include directives
  3. Taglib directives

Page Directive attributes:

JSP page directive defines the attributes that apply to the whole JSP page.

* import attribute
* language attribute
* extends attribute
* session attribute
* isThreadSafe attribute
* isErrorPage attribute
* errorPage attribute
* contentType attribute
* autoFlush attribute
* buffer attribute

Syntax of Page directive : <%@ page attributes %>

<%@ page import="java.util.Date" %>

or

<%@ page import="java.util.Date,java.net.\*" %>

<%@include file="myJSP.jsp"%>

<%@taglib uri ="taglibURI" prefix="tag prefix"%>

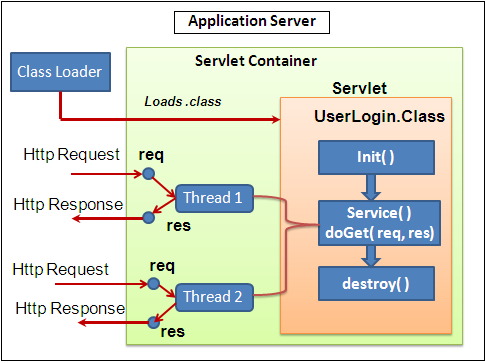
**JSP implicit objects** are created by JSP Engine during translation phase (while translating JSP to Servlet).

They are being created inside service method so we can directly use them within [**Scriptlet**](https://beginnersbook.com/2013/05/jsp-tutorial-scriptlets/) without initializing and declaring them.

There is total **9 implicit objects available in JSP**.

## Implicit Objects and their corresponding classes:

|  |  |
| --- | --- |
| **IMPLICIT OBJECT** | **CLASS** |
| Out | javax.servlet.jsp.JspWriter |
| Request | javax.servlet.http.HttpServletRequest |
| Response | javax.servlet.http.HttpServletResponse |
| Session | javax.servlet.http.HttpSession |
| Application | javax.servlet.ServletContext |
| Exception | javax.servlet.jsp.JspException |
| Page | java.lang.Object |
| pageContext | javax.servlet.jsp.PageContext |
| Config | javax.servlet.ServletConfig |



|  |  |
| --- | --- |
| Servlet | JSP |
| Servlets are faster as compared to JSP, as they have a short response time. | JSP is slower than Servlets, as the first step in the JSP lifecycle is the conversion of JSP to Java code and then the compilation of the code. |
| Servlets are Java-based codes. | JSP are HTML-based codes. |
| Servlets are harder to code, as here, the HTML codes are written in Java. | JSPs are easier to code, as here [Java](https://www.upgrad.com/blog/java-architecture-components-explained-2020/) is coded in HTML. |
| In an MVC architecture, Servlets act as the controllers. | In MVC architectures, the JSPs act as a view to present the output to the users. |
| The service() function can be overridden in Servlets. | The service() function cannot be overridden in JSPs. |
| The Servlets are capable of accepting all types of protocol requests. | The JSPs are confined to accept only the HTTP requests. |
| Modification in Servlets is a time-consuming and challenging task, as here, one will have to reload, recompile, and then restart the servers. | Modification is easy and faster in JSPs as we just need to refresh the pages. |
| Servlets require the users to enable the default sessions management explicitly, as Servlets do not provide default session management. | JSPs provide session management by default. |
| Servlets require us to implement the business logic and presentation logic in the same servlet file. | JSPs give us the flexibility to separate the business logic from the presentation logic using javaBeans. |

// JS Shopping cart e-box problem

var fs = require('fs');

var input = fs.readFileSync('input.txt').toString().trim().split("\n");

class Product {

constructor(name, price, count){

this.name = name;

this.price = parseInt(price);

this.count = parseInt(count);

this.totalcost = 0;

}

tprice(){

let fr = parseInt(this.count / 4);

//console.log(fr)

this.totalcost = this.price \* (this.count - fr);

}

static calculateTotalPrice(...ip){

let P = ip.reduce((a, b) => a+b);

return (arr.length >= 5 ? P-((10/100)\*P) : P).toFixed();

}

}

var arr = new Array();

var iprice = new Array();

for(let i=0; i<input.length; i++){

let ar = input[i].split(",");

arr.push(new Product(ar[0], ar[1], ar[2]));

arr[i].tprice();

iprice.push(arr[i].totalcost);

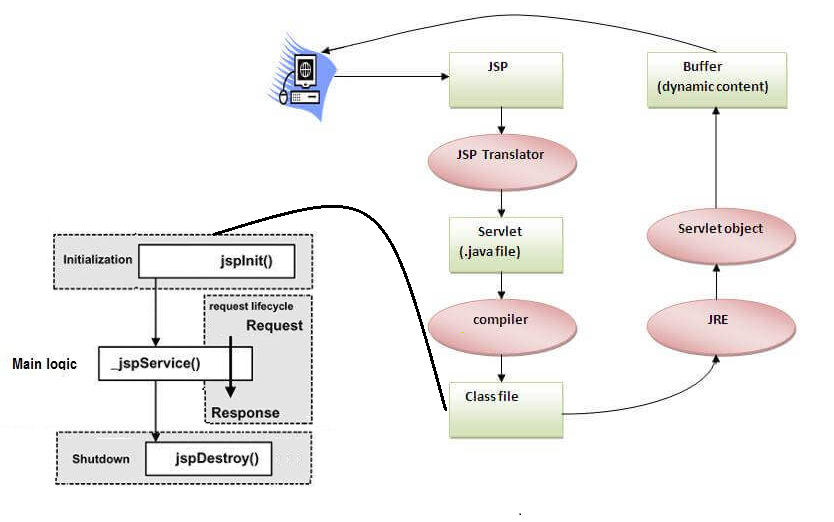
}

//console.log(arr);

//console.log(iprice)

console.log(Product.calculateTotalPrice(...iprice))

Life cycle of Jsp



Standard Action Tags:

Allow the jsp page perform some action, generate some dynamic content.

<jsp:action attributes > </jsp:action>

<jsp:action attributes />

1.<jsp:useBean >

2.<jsp:getProperty>

3.<jsp:setProperty>

4.<jsp:include>

5.<jsp:forward>

6.<jsp:param>

### 7. <jsp:plugin> Action

### 8. <jsp:body> Action

### 9. <jsp:element> Action

### 10. <jsp:text> Action

### 11. <jsp:attribute> Action

## Directives vs Actions

1. Directives are used during translation phase while actions are used during request processing phase.
2. Unlike Directives Actions are re-evaluated each time the page is accessed.

<jsp:useBean id= "instanceName" scope= "page | request | session | application"

**class**= "packageName.className" |

 <%= expression >" >

</jsp:useBean>

<jsp:setProperty name="instanceOfBean" property= "\*"   |

property="propertyName" param="parameterName"  |

property="propertyName" value="{ string | <%= expression %>}"

/>

Example:   <jsp:setProperty name="bean" property="username" value="Kumar"/>

<jsp:getProperty name="instanceOfBean" property="propertyName" />

Action Tags

<jsp:include page="Relative\_URL\_Of\_Page">

<jsp:param ... />

<jsp:param ... />

<jsp:param ... />

...

<jsp:param ... />

</jsp:include>

<jsp:forward page="display.jsp">

<jsp:param ... />

<jsp:param ... />

<jsp:param ... />

...

<jsp:param ... />

</jsp:forward>

JSTL (JSP Standard Tag Library)

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags that encapsulates the core functionality common to many JSP applications.  
  
JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

## JSTL Dependencies or Jars

In order to work with JSTL tags, we will need to add below two jars:

*javax.servlet.jsp.jstl-api-1.2.1.jar* - This contains the JSTL API interfaces and support classes. However, a large number of interfaces do not have implementation classes.

*javax.servlet.jsp.jstl-1.2.1.jar* - This contains an implementation of the JSTL API. This code implements all of the interfaces from the API above.

## Classification of The JSTL Tags

The JSTL tags can be classified according to their functions into the following JSTL tag library groups that can be used when creating a JSP page −

1. Core Tags
2. Formatting tags
3. SQL tags
4. XML tags
5. JSTL Functions

JSTL core tags provide support for iteration, conditional logic, catch an exception, URL, forward or redirect response etc.

## JSTL Core Tags List

1 *<c:out>* - Like <%= ... >, but for expressions.

2 *<c:set >* - Sets the result of an expression evaluation in a 'scope'

3 *<c:remove >* - Removes a scoped variable (from a particular scope, if specified).

4 *<c:catch>* - Catches any Throwable that occurs in its body and optionally exposes it.

5 *<c:if>* - Simple conditional tag which evalutes its body if the supplied condition is true.

6 *<c:choose>* - Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by and .

7 *<c:when>* - Subtag of that includes its body if its condition evalutes to 'true'.

8 *<c:otherwise >* - Subtag of that follows the tags and runs only if all of the prior conditions evaluated to 'false'.

9 *<c:import>* - Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.

10 *<c:forEach >* - The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .

11 *<c:forTokens>* - Iterates over tokens, separated by the supplied delimiters.

12 *<c:param>* - Adds a parameter to a containing 'import' tag's URL.

13 *<c:redirect >* - Redirects to a new URL.

14 *<c:url>* - Creates a URL with optional query parameters

## How to Use JSTL Core Tags?

To use JSTL core tags, we should include below a line of code in our JSP page:

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

## JSTL <c:forEach> Tag

*<c:forEach>* tag in JSTL is used for executing the same set of statements for a finite number of times. It’s similar to the for loop in java. This is a basic iteration tag, accepting many different collection types and supporting subsetting and other functionality.

**Syntax of <c:forEach>**

<c:forEach var="counter\_variable\_name" begin="intial\_value" end="final\_limit">

//Block of statements

</c:forEach>

<c:choose>

<c:when test="${condition1}">

//do something if condition1 is true

</c:when>

<c:when test="${condition2}">

//do something if condition2 is true

</c:when>

<c:otherwise>

//Statements which gets executed when all <c:when> tests are false.

</c:otherwise>

</c:choose>

<c:if test="${condition}">

...

..

</c:if>

The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions. The syntax used for including JSTL function library in your JSP is:

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

<hr>

<h1>JSTL fn:startsWith() function example</h1>

<c:set var="str" value="Java Guides" />

Java Guides starts with Java : ${fn:startsWith(str, 'Java')}

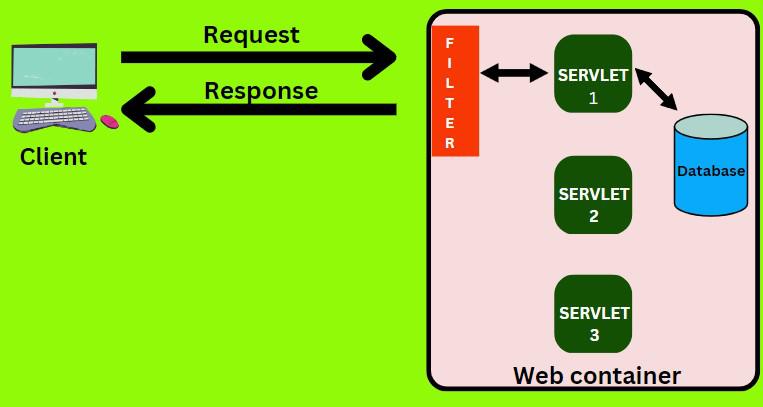
<br>

**Filters and Listeners in Servlets API**

A filter is an object that is invoked at the preprocessing and postprocessing of a request on the server, i.e., before and after the execution of a servlet for filtering the request.

Servlet Filters are **pluggable** java components that we can use to intercept and process requests before they are sent to servlets and response after servlet code is finished and before container sends the response back to the client.

Filter API (or interface) includes some methods which help us in filtering requests.



**Advantages of using filter**

* Authentication and authorization of requests for resources. (To check whether the user is valid or not and then forward its request.)
* Formatting of request body or header before sending it to the servlet. (To format the unformatted data)
* Compressing the response data sent to the client. (e.g., encrypting)
* Alter the response by adding some cookies, header information, etc.
* Input validation.

The user-defined servlet filter is pluggable, i.e., its entry is defined in the web.xml file, if we remove the entry of the filter from the web.xml file, it will be removed automatically and we don’t need to change the servlet.

**How to implement a Filter using API Filter?**

* One need to implement the following methods of the Filter API (or interface) –
  + Three methods – init(), doFilter(), destroy (). Have to override these methods. They are the lifecycle methods of a Filter.
  + doFilter will be executed in both preprocessing + postprocessing.
  + doFilter() method takes three arguments – ServletRequest, ServletResponse, FilterChain. With the help of FilterChain, we can forward the request after successful authentication.

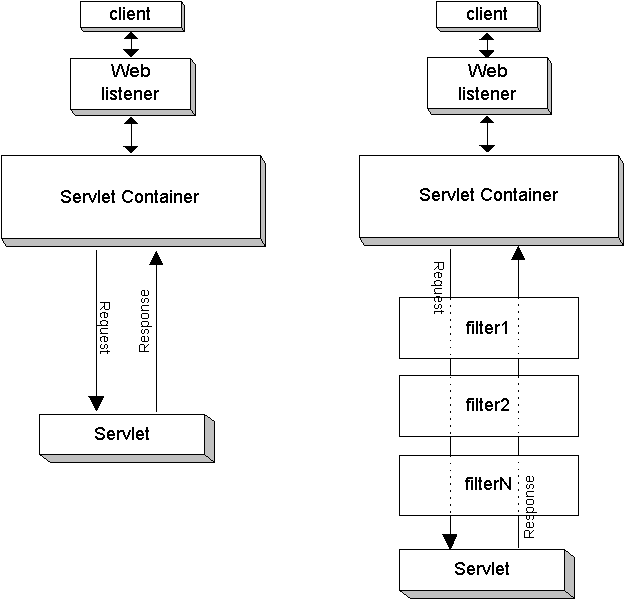
**Create a user-defined Filter class**

Create a class that implements the Filter interface and overrides all its methods, i.e., init(), doFilter(), destroy ().

If we don’t override these methods then our class will also become an abstract class.

Code written before “chain.doFilter()” will run before the servlet (preprocessing) and code written after “chain.doFilter()” will run after the servlet (post-processing).

**Configure the Filter**

* Configure this class (user-defined filter) in the web.xml.
* Map this class (user-defined filter) in the web.xml to specify when it will be executed.
  + 

## Event Listeners

Servlet Listener is used for listening to events in web containers, such as when you create a session, insert an attribute, passivate and activate in another container. The servlet container generates events that trigger the action of event listener classes. To subscribe to these events, you configure listeners in web.xml, such as HttpSessionListener.

### Event Listener Methods and Related Classes

This section contains event listener methods that are called by the servlet container when a servlet context event or session event occurs. These methods take different types of event objects as input, so these event classes and their methods as follows.

1. **javax.servlet.http.HttpSessionBindingEvent** - Events of this type are either sent to an object that implements HttpSessionBindingListener when it is bound or unbound from a session, or to a HttpSessionAttributeListener that has been configured in the web.xml when any attribute is bound, unbound or replaced in a session. The session binds the object by a call to HttpSession.setAttribute and unbinds the object by a call to HttpSession.removeAttribute. We can use this event for cleanup activities when object is removed from session.
2. **javax.servlet.http.HttpSessionEvent** - This is the class representing event notifications for changes to sessions within a web application.
3. **javax.servlet.ServletContextAttributeEvent** - Event class for notifications about changes to the attributes of the ServletContext of a web application.
4. **javax.servlet.ServletContextEvent** - This is the event class for notifications about changes to the servlet context of a web application.
5. **javax.servlet.ServletRequestEvent** - Events of this kind indicate lifecycle events for a ServletRequest. The source of the event is the ServletContext of this web application.
6. **javax.servlet.ServletRequestAttributeEvent** - This is the event class for notifications of changes to the attributes of the servlet request in an application.

Servlet API provides following Listener interfaces.

1. **javax.servlet.ServletContextListener** - Interface for receiving notification events about ServletContext lifecycle changes.
2. **javax.servlet.ServletContextAttributeListener** - Interface for receiving notification events about ServletContext attribute changes.
3. **javax.servlet.ServletRequestListener** - Interface for receiving notification events about requests coming into and going out of scope of a web application.
4. **javax.servlet.ServletRequestAttributeListener** - Interface for receiving notification events about ServletRequest attribute changes.
5. **javax.servlet.http.HttpSessionListener** - Interface for receiving notification events about HttpSession lifecycle changes.
6. **javax.servlet.http.HttpSessionBindingListener** - Causes an object to be notified when it is bound to or unbound from a session.
7. **javax.servlet.http.HttpSessionAttributeListener** - Interface for receiving notification events about HttpSession attribute changes.
8. **javax.servlet.http.HttpSessionActivationListener** - Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated. A container that migrates session between VMs or persists sessions is required to notify all attributes bound to sessions implementing HttpSessionActivationListener.

#### HttpSessionListener Methods, HttpSessionEvent Class

The HttpSessionListener interface specifies the following methods:

* void sessionCreated(HttpSessionEvent hse)

The servlet container calls this method to notify the listener that a session was created.

* void sessionDestroyed(HttpSessionEvent hse)

The servlet container calls this method to notify the listener that a session was destroyed.

* HttpSession getSession()
* Uisng this object you extract the information of client specific information during the session,

Security Issues In Servlets

Servlets, being a fundamental part of Java web development, are generally secure when implemented correctly. However, like any technology, there are potential security concerns to be aware of:

1. \*\*Injection Attacks:\*\* Servlets might be vulnerable to various injection attacks, including SQL injection, where malicious SQL queries are injected into input fields. Developers should use prepared statements or parameterized queries to prevent these attacks.

2. \*\*Cross-Site Scripting (XSS):\*\* Servlets can be susceptible to XSS attacks if they render user inputs without proper validation or escaping. Input validation and output encoding are essential to mitigate this risk.

3. \*\*Cross-Site Request Forgery (CSRF):\*\* Servlets can be prone to CSRF attacks if they don’t implement proper anti-CSRF tokens or mechanisms. These attacks trick users into performing actions they didn’t intend to do.

4. \*\*Session Management:\*\* Improper session management in servlets can lead to session hijacking or fixation. Developers need to ensure secure session handling, use secure cookies, and avoid passing sensitive information in URLs.

5. \*\*File Upload Security:\*\* Servlets handling file uploads can be vulnerable to attacks if proper validation and file type checking aren't implemented. This can lead to uploading malicious files or overloading the server with large files.

6. \*\*Authentication and Authorization:\*\* Weak authentication and authorization mechanisms can lead to unauthorized access to resources. It’s crucial to implement strong authentication methods and proper access controls.

7. \*\*Error Handling and Information Leakage:\*\* Inadequate error handling can expose sensitive information, providing attackers with insights into the system's architecture or vulnerabilities. Servlets should handle errors gracefully without exposing internal details.

8. \*\*Denial of Service (DoS) Attacks:\*\* Servlets can be susceptible to DoS attacks if not properly configured to handle large or unexpected amounts of traffic. This includes limiting resource consumption and setting appropriate timeouts.

To mitigate these security risks in servlets, developers should follow best practices such as input validation, output encoding, implementing secure authentication mechanisms, using HTTPS, keeping software up-to-date, and regularly conducting security audits and testing. Additionally, leveraging security frameworks and libraries specifically designed for web application security can bolster the protection of servlet-based applications.

Securing servlets involves implementing various measures to protect the application from potential vulnerabilities and threats. Here's a guide on how to provide security to servlets:

### 1. Input Validation:

- \*\*Sanitize Inputs:\*\* Filter and validate all user inputs to prevent injection attacks like SQL injection, XSS, and other vulnerabilities. Use validation libraries or custom validation methods to ensure data integrity.

### 2. Authentication and Authorization:

- \*\*Implement Secure Authentication:\*\* Use strong authentication mechanisms like OAuth, JWT, or multi-factor authentication to verify user identities.

- \*\*Access Control:\*\* Enforce proper authorization and access control based on user roles and permissions. Servlets should check user credentials and rights before granting access to resources.

### 3. Session Management:

- \*\*Secure Session Handling:\*\* Implement secure session management techniques. Use secure cookies, regenerate session IDs on login, and avoid passing sensitive information in URLs.

- \*\*Session Timeout:\*\* Set appropriate session timeouts to prevent session hijacking or fixation.

### 4. Secure Communications:

- \*\*HTTPS Usage:\*\* Ensure that all communications between the client and server occur over HTTPS to encrypt data transmission and prevent eavesdropping or tampering.

### 5. Error Handling:

- \*\*Custom Error Handling:\*\* Implement custom error pages and avoid displaying detailed error messages to users. Log errors for developers without revealing sensitive information to users.

### 6. File Upload Security:

- \*\*Validate File Types and Size:\*\* Perform strict validation on file uploads, including checking file types, limiting sizes, and ensuring proper file handling to prevent potential security risks.

### 7. Secure Configuration:

- \*\*Server Configuration:\*\* Secure the servlet container or application server by configuring it according to best security practices. Disable unnecessary features, ports, or services.

### 9. Use Security Frameworks:

- \*\*Leverage Security Libraries:\*\* Utilize security-focused libraries and frameworks like Spring Security, Apache Shiro, or OWASP libraries that offer robust security features and help in mitigating common security risks.