INSY-413

WEB TECHNOLOGY & INTERNET

Theoretical and practical course, 4 Credits

Blessed assurance, Jesus is mine
O what a foretaste of glory divine
Heir of salvation, purchase of God
Born of His Spirit, washed in His blood

This is my story, this is my song
Praising my Savior all the day long
This is my story, this is my song
Praising my Savior all the day long

COURSE CONTENT

⇒ **Prerequisite: Java Programming, Web Design**

- Understanding the Internet, WWW and Web Application
- Servlets Technology
- Java Server Pages (JSP) Technology
- Spring MVC
- Web Services
- Mini Project

Module 1 – Understanding Internet ,WWW, and Web Application

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- Describe the Internet and its history and Examine Internet Protocol Suite
- Explain the World Wide Web, HTTP, WEB Browsers, DNS, URL, Port
- Explain WEB Servers, Application Servers and
- Understand Web Application Building Block
- Implementing a simple Web Application

Module 2 – Servlet technology

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- Create dynamic web applications using Java Servlets to handle HTTP requests and generate dynamic content.
- Comprehend the servlet lifecycle, including initialization, request handling, and destruction phases.
- Efficiently handle HTTP GET and POST requests, read request parameters, and send appropriate responses to clients.
- Implement session management techniques, use session attributes, and understand how cookies and URL rewriting work.
- Process HTML forms in servlets, read form data, validate user input, and generate dynamic HTML forms.
- Create and use servlet filters to perform tasks like request/response modification, authentication, and logging.
- Handle exceptions in servlets, configure custom error pages, and provide meaningful error messages to users.

Module 3 – Java Server Pages

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- Develop web pages that can generate dynamic content by embedding Java code within JSP files.
- Comprehend and apply JSP syntax elements, including scriptlets, expressions, declarations, comments, and directives.
- Employ the JavaServer Pages Standard Tag Library (JSTL) to perform common web development tasks such as loops, conditionals, and data display in a more structured and efficient manner.
- Process and validate HTML form submissions in JSP, extract form data, and provide feedback to users.
- Implement session management techniques in JSP, including storing and retrieving session attributes and handling cookies and URL rewriting.
- Connect to databases using JDBC and execute SQL queries within JSP to retrieve, display, and manipulate data.
- Understand and apply the Model-View-Controller (MVC) design pattern in web development, separating business logic from presentation using JSP.

Module 4 – Spring MVC

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- ❏ Explain how to develop web applications following the Model-View-Controller (MVC) architectural pattern using Spring MVC.
- ❏ Comprehend the key components and request flow in a Spring MVC application, including controllers, views, and model attributes.
- ❏ Create controller classes to handle HTTP requests, including request mapping, path variables, and different HTTP methods (GET, POST, etc.).
- ❏ Integrate views with Spring MVC, using template engines like Thymeleaf, and understand data binding and form handling.
- ❏ Create model classes and bind data between views and controllers, including handling form submissions, data conversion, and validation.
- ❏ Implement request and response interceptors, handle exceptions globally, and write custom filters for tasks like security and internationalization.
- ❏ Configure authentication and authorization using Spring Security for web application security.
- ❏ Build RESTful APIs using Spring MVC, consume RESTful services, and understand versioning and documentation.
- ❏ Handle file uploads and implement file download functionality in Spring MVC applications.
- ❏ Create Spring Boot-based Spring MVC applications, leveraging auto-configuration and development tools for efficient development.
- ❏ Write unit tests for Spring MVC controllers, use MockMvc for integration testing, and test validation and error handling.
- ❏ Collaborate effectively with team members on web development projects, understand project requirements, and build complete web applications.

Internet

- Internet is a short form of the technical term *internetwork*, the result of interconnecting computer networks with special gateways or routers.
- The Internet is a global network of interconnected computers and servers that communicate with each other through standardized protocols.
- It enables the exchange of information, data, and communication between devices worldwide, providing a platform for various services such as email, online shopping, social media, and more.
- It operates without a central governing body.
- However, to maintain interoperability, all technical and policy aspects of the underlying core infrastructure and the principal name spaces are administered by the Internet Corporation for Assigned Names and Numbers (ICANN).

History of Internet

- The Internet has its roots in the **1960s** when the U.S. Department of Defense created ARPANET, a precursor to the modern Internet.
- Over the decades, the Internet evolved through the development of new technologies, such as the **World Wide Web, email, and the rise of mobile devices.**
- **In the 1990s, the commercialization of the Internet began**, leading to an explosion of growth and the development of e-commerce, social media, and other online services.
- Today, the Internet is a vital tool for communication, commerce, and information exchange, with billions of people accessing it globally.

World Wide Web

- WWW is a system of interlinked hypertext documents accessed via the Internet.
- The World Wide Web, or simply **Web**, is a way of accessing information over the medium of the Internet.
- It is an information-sharing model that is built on top of the Internet.
- **The Web uses the HTTP protocol,** only one of the languages spoken over the Internet, **to transmit data.**
- It was created in 1989 by British computer scientist Tim Berners-Lee, who envisioned a way for people to share information and access it easily from anywhere in the world.

World Wide Web Consortium (W3C)

- The World Wide Web Consortium (W3C) is an international standards organization founded in 1994 by Tim Berners-Lee, the inventor of the World Wide Web.
- Its mission is to lead the Web to its full potential by developing and promoting open standards that ensure its long-term growth and interoperability.
- The W3C develops technical standards, such as HTML, CSS, and JavaScript, and provides guidelines and tools to improve accessibility and the user experience on the Web.
- It also fosters collaboration between members, which include governments, corporations, and individuals, to advance the Web as a platform for innovation and creativity.

Web Page

- A web page is a **document** or information resource that is **suitable for the World Wide Web** and can be accessed through a web browser and displayed on a monitor or mobile device.
- This information is usually in HTML or XHTML format and may provide navigation to other web pages via hypertext links.
- Web pages frequently include other resources such as style sheets, scripts, and images in their final presentation.
- Web pages are requested and served from web servers using Hypertext Transfer Protocol (HTTP).
- Web pages may consist of files of static text and other content stored within the web server's file system (**static web pages**), or may be constructed by server-side software when they are requested (**dynamic web pages**).

Website

- A website is a collection of related web pages containing images, videos, and other digital assets that are hosted on a single server and can be accessed through a single domain name.
- All publicly accessible websites collectively constitute the World Wide Web.
- Web sites can be static or dynamic.
- A dynamic website is a type of website that generates and displays different content each time it is accessed, based on user input or changes to the database. This is in contrast to a static website, which displays the same content every time it is visited.
- Dynamic websites are often used for e-commerce, social media, and other applications that require user interaction and the ability to update content in real-time. They are built using server-side technologies, such as PHP, Java, Python, Ruby on Rails, or ASP.NET, that enable the server to process user requests and generate customized content on-the-fly.

Assignment #1. Understanding Web Applications

- Create a Simple Human Resource Web Application
  - Create a landing page
  - Create the employee registration page (the form should have at least 10 fields)
  - a login page

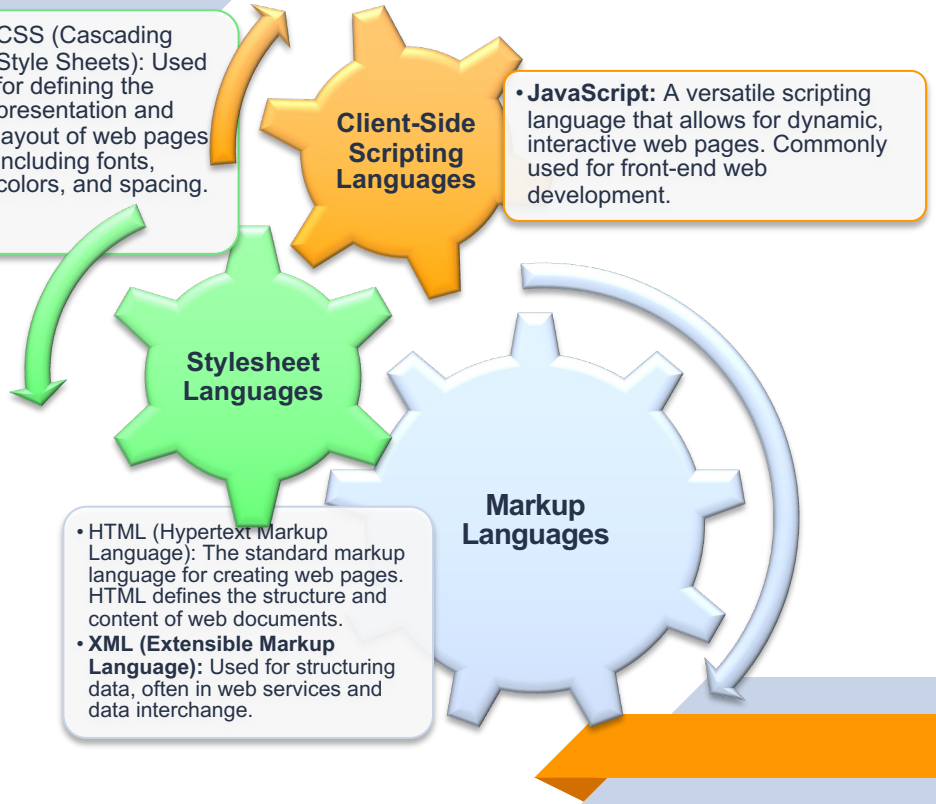
Use HTML, style with CSS, and validate the forms with JavaScript.

Web technology

Web technology encompasses a wide range of tools, frameworks, languages, and protocols used for building and maintaining websites and web applications.

Web Technology Stack

A web technology stack, also known as a tech stack or software stack, refers to the **combination** of **programming languages, frameworks, libraries, and tools** used to build and run a web application or website.



Web Technology Stack

Server-Side Scripting Languages

- **PHP**: A popular server-side scripting language for web development.
- **Python** (e.g., Django, Flask): Used for building web applications and websites.
- **Ruby** (Ruby on Rails): Known for its simplicity and rapid development capabilities
- **Java** (e.g., Java Servlets, JSP): Used for building enterprise-level web applications.

Web Frameworks

- **JavaScript (Front-End)** e.g., React, Angular, Vue.js
- **Python e.g.,** Django, Flask
- **PHP** (e.g., Laravel, Symfony)
- **Java** (e.g., JSF, Spring MVC)
- **Others:**
  - Ruby on Rails (Rails):
  - ASP.NET (C#)
  - Go (Golang), etc.

Databases

- SQL Databases (e.g., MySQL, PostgreSQL, SQL Server): Relational databases commonly used for web applications.
- NoSQL Databases (e.g., MongoDB, Cassandra): Non-relational databases often used for storing and retrieving large volumes of data.

Web Servers

- **Apache HTTP Server:** A widely used open-source web server.
- **Nginx:** Known for its performance and scalability in serving web content.
- **Microsoft Internet Information Services (IIS):** Microsoft's web server solution for Windows environments.

APIs

- **REST (Representational State Transfer):** A style of architectural principles for designing networked applications, commonly used for web services.
- **GraphQL:** A query language for APIs, designed to provide more flexibility and efficiency in data retrieval.

Content Management Systems (CMS)

- **WordPress:** A popular open-source CMS for creating websites and blogs.
- **Joomla, Drupal, TYPO3:** Other CMS options for various needs.

Web Security

- **HTTPS:** A secure version of HTTP that encrypts data transmitted between the client and server.
- **SSL/TLS (Secure Sockets Layer/Transport Layer Security):** Protocols for securing data in transit.
- **Web Application Firewalls (WAFs):** Security measures to protect web applications from attacks.

Web Development Tools

- Version Control Systems (eg., Git)
- IDEs (Integrated Development Environments)
- Web development Browsers (e.g., Google Chrome DevTools)

Servlets Technology

Servlets

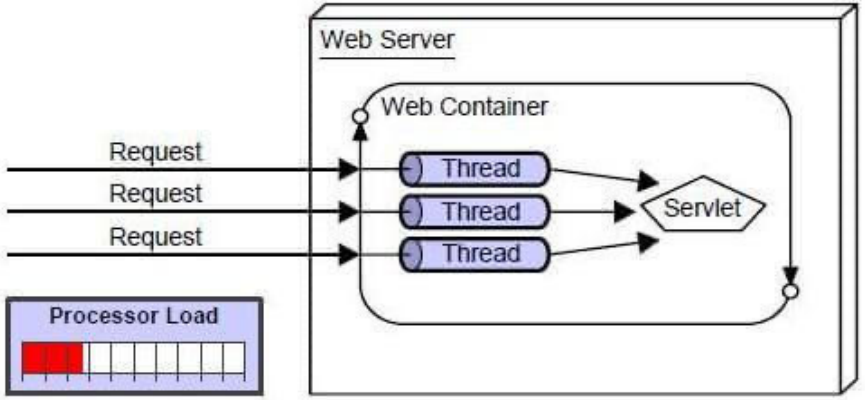
- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- **A Servlet is a Java class** that is **used to extend** the capabilities of servers that host applications accessed by means of a request-response programming model.
- The **server-side extensions** are nothing but the technologies that are used to create dynamic Web pages.
- Servlets can be used to process HTTP requests, handle form submissions, manage sessions, and perform other server-side tasks.
- Servlets are part of Java Enterprise Edition (Java EE) and are **standard technology for building web-based applications**.

Servlets (Cnt'd)

- **Servlet** technology is robust and scalable because of Java language.
- Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
- However, there were many disadvantages to CGI technology:
  - If the number of clients increases, it **takes more time for sending the response**.
  - **For each request, it starts a process**, and the web server is limited to start processes.
  - It uses **platform-dependent language** e.g. C, C++, perl.

Advantages of Servlets

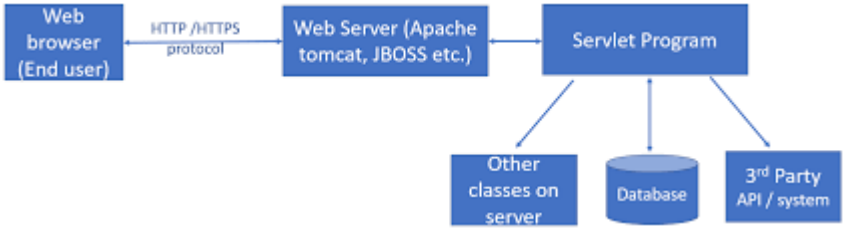
- **Better performance:** because it creates a thread for each request, not process.
- **Portability:** because it uses Java language.
- **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
- **Secure:** because it uses Java language.



Servlet Architecture

Execution of Servlets basically involves six basic steps:

- The clients send the request to the webserver.
- The web server receives the request.
- The web server passes the request to the corresponding servlet.
- The servlet processes the request and generates the response in the form of output.
- The servlet sends the response back to the webserver.
- The web server sends the response back to the client and the client browser displays it on the screen.



The Servlet Container

- **Servlet container**, also known as **Servlet engine** is an integrated set of objects that provide a run time environment for Java Servlet components.
- In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

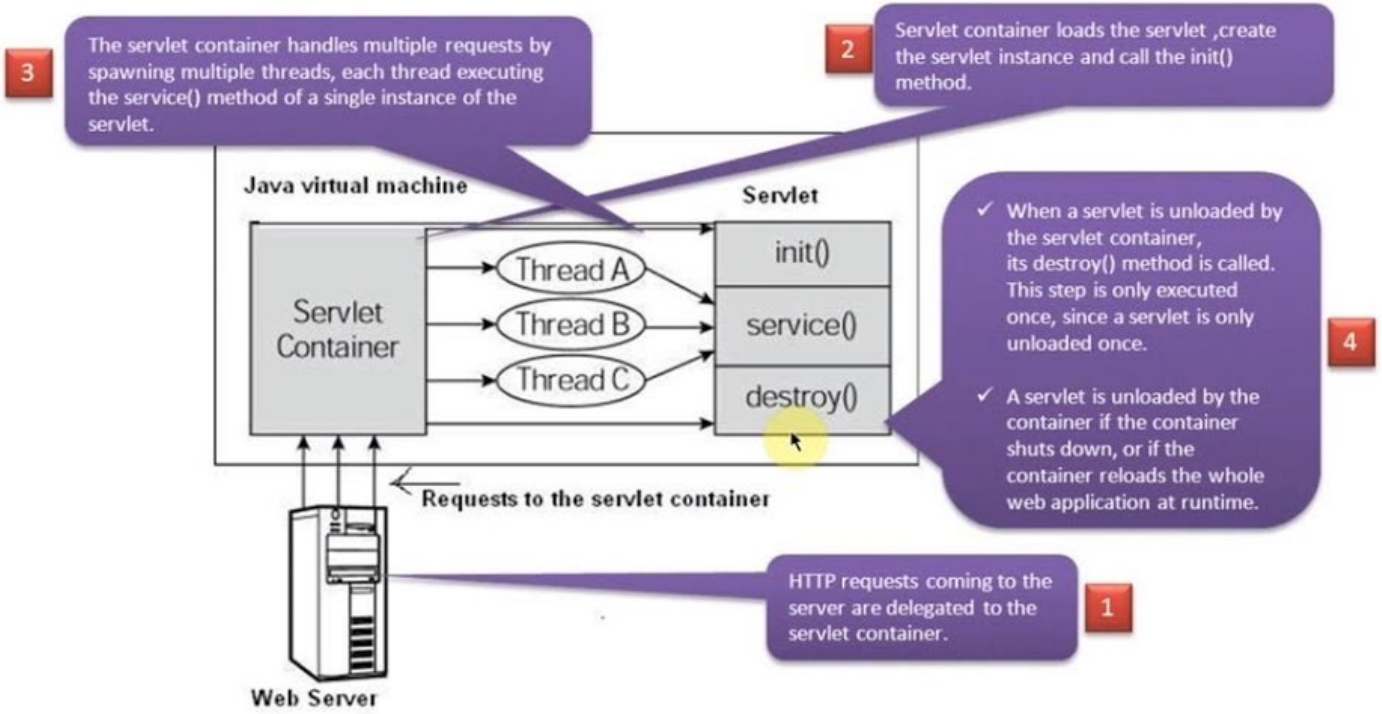
The Servlet Container

Services provided by the Servlet container :

- **Network Services:** Loads a Servlet class. The loading may be from a local file system, a remote file system or other network services. The Servlet container provides the network services over which the request and response are sent.
- **Decode and Encode MIME**(Multipurpose Internet Mail Extensions)-**based messages:** Provides the service of decoding and encoding MIME-based messages.
- **Manage Servlet container:** Manages the lifecycle of a Servlet.
- **Resource management** Manages the static and dynamic resources, such as HTML files, Servlets, and JSP pages.
- **Security Service:** Handles authorization and authentication of resource access.
- **Session Management:** Maintains a session by appending a **session ID** to the URL path.

Servlets & Jsp

How Servlets works



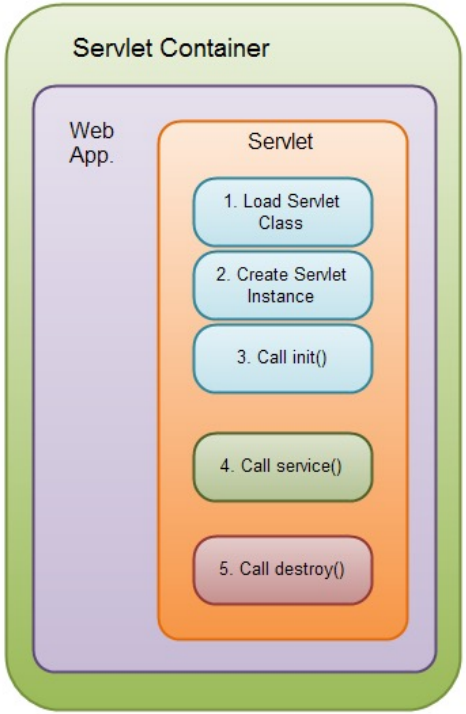
Servlet API Interfaces and Classes

- The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

javax.servlet Package	javax.servlet.http Package
Interfaces: Servlet ServletConfig ServletContext ServletRequest ServletResponse RequestDispatcher Filter FilterChain FilterConfig ServletRequestListener ServletRequestAttributeListener ServletContextListener ServletContextAttributeListener	**Interfaces:** HttpServletRequest HttpServletResponse HttpSession HttpSessionListener HttpSessionAttributeListener HttpSessionBindingListener HttpSessionActivationListener
Classes: GenericServlet ServletRequestWrapper ServletResponseWrapper ServletInputStream ServletOutputStream ServletContextEvent ServletContextAttributeEvent ServletRequestEvent ServletRequestAttributeEvent ServletException UnavailableException	**Classes:** Cookie HttpServlet HttpServletRequestWrapper HttpServletResponseWrapper HttpSessionEvent HttpSessionBindingEvent

Methods of Servlet interface

- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly).
- It provides 3 life cycle methods that are used to **initialize** the servlet, to **service** the requests, and to **destroy** the servlet and 2 non-life cycle methods:
  - **getServletConfig()** returns the object of ServletConfig
  - **getServletInfo()** returns information about servlet such as writer, copyright, version etc.

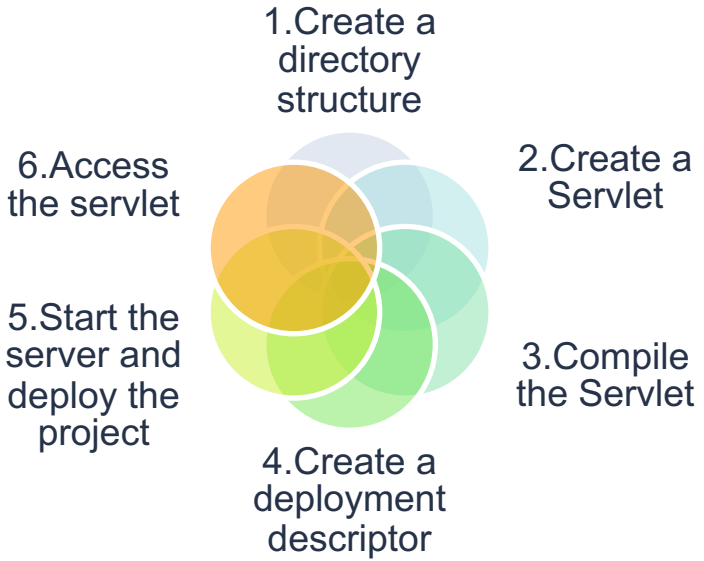


Steps to create a servlet

There are given 6 steps to create a **servlet example**. These steps are required for all the servers. The servlet example can be created by three ways:

- By implementing Servlet interface,
- By inheriting GenericServlet class, (or)
- By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.



Notes on GET and POST Methods

GET Method

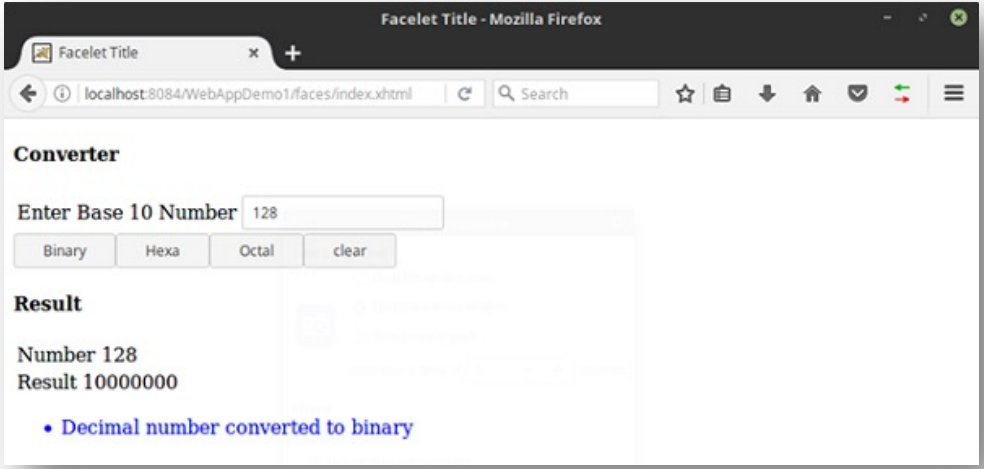
Appends form-data into the URL in name/value pairs	The length of a URL is limited (about 3000 characters)	Never use GET to send sensitive data! (will be visible in the URL)	Useful for form submissions where a user wants to bookmark the result	GET is better for non-secure data, like query strings in Google

POST Method

Appends form-data inside the body of the HTTP request (data is not shown in URL)	Has no size limitations	Form submissions with POST cannot be bookmarked

Exercise. Understanding Web Applications

- Using Servlets create a Simple Number Converter as shown by the sample page (right)
- Create a Servlet Class to handle the Student Admission request and return the preview of entered information.
- Implement the Authentication using servlets.



Servlets - Http Status Codes

- HTTP status codes are three-digit codes returned by an HTTP server in response to a client's request. They **indicate** whether a specific **HTTP request** has been successfully completed or not.
- Here are some of the most common HTTP status codes that you may encounter when working with Servlets:

Types of HTTP Status Codes

- 100s: Informational codes: the server acknowledges the request initiated by the browser and that it is being processed (100–199).
- 200s: Success codes: request received, understood, processed and expected info relayed to browser (200–299).
- 300s: Redirection codes: a different destination has been substituted for the requested resource; further action by the browser may be required (300–399).
- 400s: Client error codes: website or page not reached; page unavailable or there was a technical problem with the request (400–499).
- 500s: Server error codes: request was accepted, but due to an error the server could not fulfil the request (500–599).

Some of the most common HTTP status codes

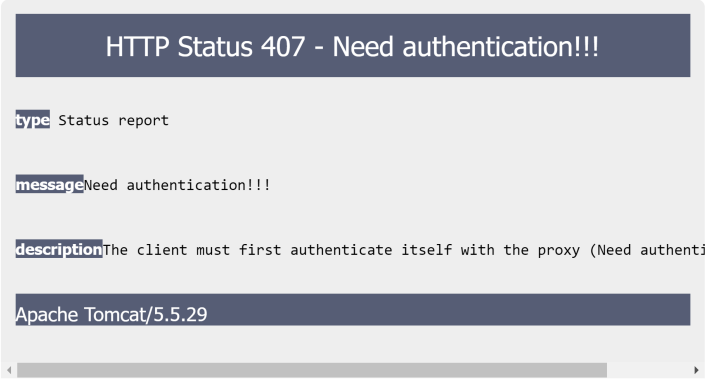
- 200 OK: The request was successful and the server has returned the requested data.
- 201 Created: The request was successful and a new resource has been created on the server.
- 204 No Content: The request was successful, but there is no representation to return (i.e., the response is empty).
- 400 Bad Request: The request could not be understood or was missing required parameters.
- 401 Unauthorized: The request requires user authentication.
- 403 Forbidden: The user is authenticated but does not have access to the requested resource.
- 404 Not Found: The requested resource could not be found on the server.
- 500 Internal Server Error: An error occurred on the server.

Methods to Set HTTP Status Code

The following methods can be used to set HTTP Status Code in the servlet program. These methods are available with *HttpServletResponse* object.

- public void setStatus (int statusCode)
- public void sendRedirect(String url)
- public void sendError(int code, String message)

Eg. response.sendError(407, "Need authentication!!!");



Servlets - Exception Handling

- When a servlet throws an exception, the web container searches the configurations in **web.xml** that use the *exception-type element* for a match with the thrown exception type.
- Use the **error-page** element in web.xml to specify the invocation of servlets in response to certain **exceptions** or HTTP **status codes**.

```
<!-- error-code related error pages -->
<error-page>
   <error-code>404</error-code>
   <location>/ErrorHandler</location>
</error-page>

<error-page>
   <error-code>403</error-code>
   <location>/ErrorHandler</location>
</error-page>

<!-- exception-type related error pages -->
<error-page>
   <exception-type>
      javax.servlet.ServletException
   </exception-type >
   <location>/ErrorHandler</location>
</error-page>

<error-page>
   <exception-type>java.io.IOException</exception-type >
   <location>/ErrorHandler</location>
</error-page>
```

Servlets Filters

Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:

- To intercept client requests before accessing a resource at back end.
- To manipulate responses from server before they are sent back to the client.

Filters Types

- Authentication Filters.
- Data compression Filters.
- Encryption Filters.
- Filters that trigger resource access events.
- Image Conversion Filters.
- Logging and Auditing Filters.
- MIME-TYPE Chain Filters.
- Tokenizing Filters .
- XSL/T Filters That Transform XML Content.

Servlets Filters Deployment

- Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet names or URL patterns in your application's deployment descriptor.
- When the web container starts up the web application, it creates an instance of each filter that has been declared in the deployment descriptor.

A filter is simply a Java class that implements the javax.servlet.Filter **interface**. The javax.servlet.Filter interface defines three methods:

- **public void init(FilterConfig)**
- **public void doFilter (ServletRequest, ServletResponse, FilterChain)**
- **public void destroy()**

Servlets - Cookies

- Cookies are small text files that are stored on a client's computer by a website's server.
- They are used to store information such as user preferences, shopping cart contents, login status, etc.
- By remembering user preferences and login status, cookies can improve the overall experience for the user by reducing the amount of data they need to enter on subsequent visits.
- Cookies should not be used to store sensitive information, as they can be accessed and potentially compromised by malicious actors.

```java
public class CookieExampleServlet extends HttpServlet {

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

    // Check if a cookie with the name "visited" exists
    Cookie visitedCookie = null;
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
      for (Cookie cookie : cookies) {
        if (cookie.getName().equals("visited")) {
          visitedCookie = cookie;
          break;
        }
      }
    }

    // If a cookie does not exist, create one and set its value to "yes"
    if (visitedCookie == null) {
      visitedCookie = new Cookie("visited", "yes");
      response.addCookie(visitedCookie);
      response.getWriter().println("This is your first visit.");
    } else {
      response.getWriter().println("Welcome back! You last visited on " + visitedCookie.getValue());
    }
  }
}
```

Servlets - Session Tracking

- HTTP is a "**stateless**" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
- Three ways to maintain session between web client and web server are:
  - **Cookies**: A webserver can assign a unique session ID as a cookie to each web client. (**Cons:** some browsers do not support cookies)
  - **Hidden Form Fields**: A web server can send a hidden HTML form field along with a unique session ID. (**Cons**: regular (<A HREF...>) hypertext link does not result in a form submission)
  - **URL Rewriting:** append some extra data on the end of each URL that identifies the session (**Cons**: have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page)

Servlets - **HttpSession**

- The **HttpSession** interface in Java Servlets is used to represent a session, which is a mechanism for tracking user interactions with a web application across multiple requests.
- The session is identified by a unique session ID, which is sent to the client in the form of a cookie.
- When the client sends subsequent requests, this session ID is sent back to the server, allowing the server to identify the user's session and retrieve the appropriate information.

Servlets - **HttpSession**

```java
// Creating a session
HttpServletRequest request;
HttpSession session = request.getSession();
session.setAttribute("username", "John Doe");

// Accessing the session
HttpServletRequest request;
HttpSession session = request.getSession();
String username = (String) session.getAttribute("username");
```

It's important to note that the session information is stored on the server and is not accessible to the client. This helps to keep sensitive information, such as the user's password, secure.

- The **HttpSession** interface provides methods for storing and retrieving information in the session, such as setAttribute and getAttribute.
- Additionally, it provides methods for managing the session, such as invalidate to end the session, and getCreationTime, and getLastAccessedTime to retrieve the time when the session was created and last accessed.

Servlet Listeners

- In Java Servlets, listeners are components that listen for specific events that occur in the Servlet environment, such as
  - the creation or destruction of the Servlet context,
  - the creation or destruction of a session, or
  - the start or completion of a request.
- Listeners are implemented as Java classes that implement one of the Servlet listener interfaces, such as ServletContextListener, HttpSessionListener, or ServletRequestListener.
- They are commonly used for tasks such as logging, tracking user sessions, or performing cleanup when the context is destroyed.

ServletContextListener

The ServletContextListener interface is used to listen for events related to the Servlet context, such as when the context is *created* or *destroyed*.

For example, you might use this interface to perform tasks such as

- initializing resources,
- setting up a database connection, or
- cleaning up resources when the context is destroyed.

```
public class MyContextListener implements ServletContextListener {
  public void contextInitialized(ServletContextEvent sce) {
      System.out.println("Servlet context initialized");
  }
  public void contextDestroyed(ServletContextEvent sce) {
      System.out.println("Servlet context destroyed");
  }
}
```

HttpSessionListener

- **The HttpSessionListener interface** is used to listen for events related to the creation and destruction of a session.
- For example, you might use this interface to keep track of the number of active sessions in a web application.

```
public class MySessionListener implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("Session created");
    }
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("Session destroyed");
    }
}
```

ServletRequestListener

- **The ServletRequestListener interface** is used to listen for events related to the processing of a request, such as when a request is received or when the response is about to be sent.
- For example, you might use this interface to keep track of the number of active requests in a web application.

```
public class MyRequestListener implements ServletRequestListener {
  public void requestInitialized(ServletRequestEvent sre) {
      System.out.println("Request received");
  }
  public void requestDestroyed(ServletRequestEvent sre) {
      System.out.println("Request processed");
  }
}
```

Servlets - Annotations

- In Java Servlets, annotations are a convenient way to **declare Servlet components, such as Servlets, filters, and listeners**, without the need for configuration in the web.xml file.
- Annotations were introduced in Servlet API version 3.0 and are now a standard way of declaring Servlet components in modern Java web applications.
- The annotation types introduced in Servlet 3.0 are –

@WebServlet	**@WebInitParam**	**@WebFilter**
@WebListener	**@HandlesTypes**	**@HttpConstraint**
@HttpMethodConstraint	**@MultipartConfig**	**@ServletSecurity**

Commonly used Servlet annotations

@WebServlet: This annotation is used to declare a Servlet component. It can be used to specify the URL mapping for the Servlet and to set various Servlet initialization parameters.

```
@WebServlet(urlPatterns = "/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.getWriter().println("Hello, World!");
    }
}
```

Commonly used Servlet annotations (Cont'd)

@WebFilter: This annotation is used to declare a filter component. It can be used to specify the URL mapping for the filter and to set various filter initialization parameters.

```
@WebFilter(urlPatterns = "/*")
public class LoggingFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
        System.out.println("Request received");
        chain.doFilter(request, response);
        System.out.println("Response sent");
    }
}
```

Commonly used Servlet annotations (Cont'd)

@WebListener: This annotation is used to declare a listener component. It can be used to register an event listener for the Servlet context, session, or request.

```
@WebListener
public class MyListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("Servlet context initialized");
    }
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("Servlet context destroyed");
    }
}
```

Assignment #3. Servlet

- To the previous exercise, add a SignUp page (email, and password). Once signed up the user will be redirected to the login page.
- The user must be logged in to have access to the student admission form (Apply authentication filters and Session management).
- To the Student admission Page, add upload actions:
  - Student passport picture. *Supported types*: .jpg or .png
  - Certificates (Diploma). *Supported type*: only PDF format
- Using **JavaMail API** send submission confirmation email to the candidate, once the admission request is submitted.
- Internationalize the application: 3 languages: **English, French and any Local language**

Java Server Pages (JSP) Technology

Java Server Pages

- JSP stands for Java Server Pages. It is a server-side technology used for creating dynamic web content.
- JSP allows the integration of Java code and scripting elements into web pages.
- JSP files are similar to HTML files but contain special tags and scripting elements that allow dynamic content to be generated.
- When a JSP file is requested by a client, the server processes the file and generates HTML code that is sent to the client's web browser.
- JSP technology is often used in combination with servlets, which are Java classes that handle requests and responses between the server and the client.
- *Together, JSP and servlets provide a powerful platform for building dynamic, data-driven web applications using Java technology.*

Java Server Pages

In brief,

- JSP is a Web-based technology that helps us to create dynamic and platform-independent web pages.
- JSP is an advanced version of Servlet Technology
- JSP tags are used to insert JAVA code into HTML pages.
- In this, Java code can be inserted in HTML/ XML pages or both
- JSP is first converted into servlet by JSP container before processing the client's request.

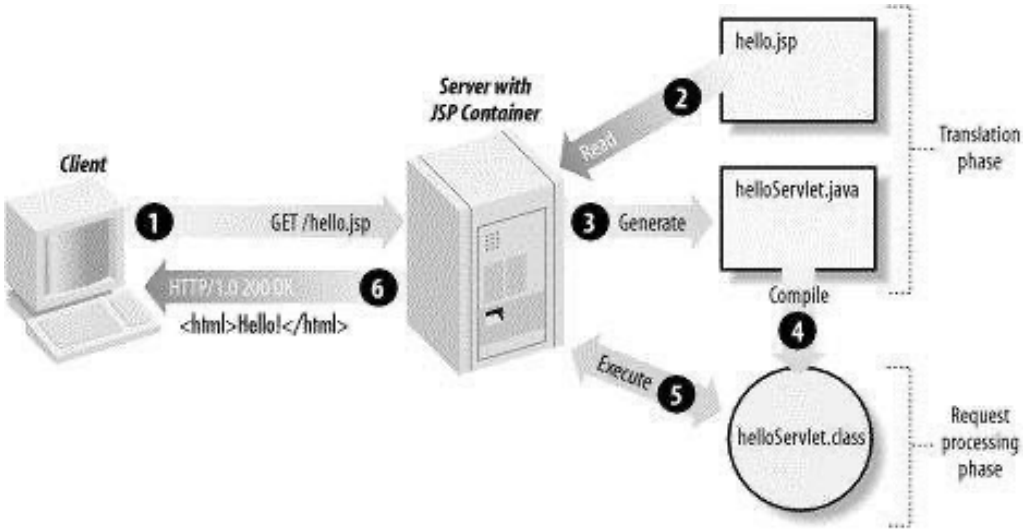
Features of JSP

- **Coding in JSP is easy** :- As it is just adding JAVA code to HTML/XML.
- **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
- **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
- **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.

Features of JSP (Cont'd)

- **Portable, Powerful, flexible, and easy to maintain**:- as these are browser and server independent.
- **No Redeployment and No Re-Compilation**: It is dynamic, secure, and platform-independent so no need to re-compile.
- **Extension to Servlet**:- as it has all features of servlets, implicit objects, and custom tags

JSP Architecture



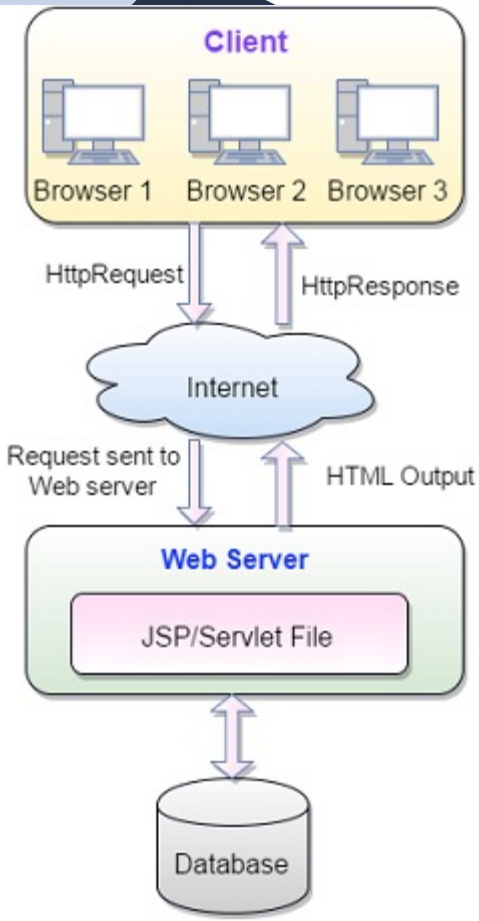
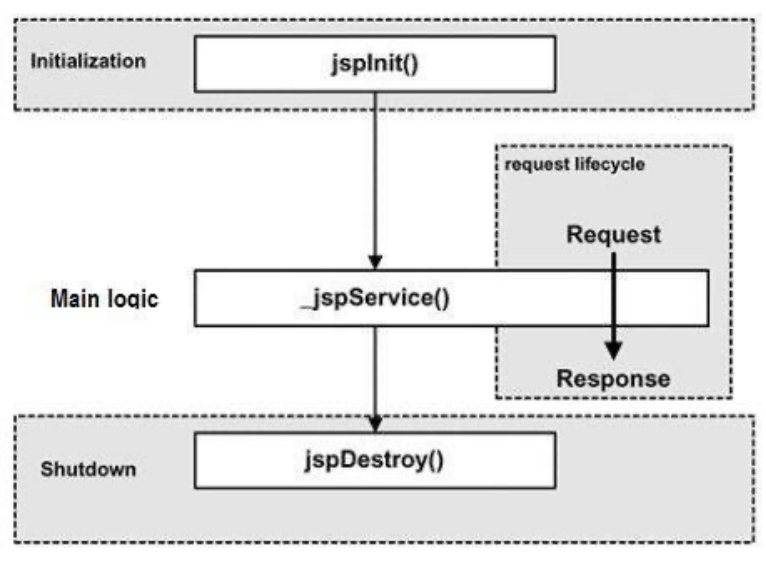


Fig: JSP Architecture

JSP - Lifecycle

- A JSP life cycle is defined as the process from its creation till the destruction.
- This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.
- The following are the paths followed by a JSP –
  - Compilation
  - Initialization
  - Execution
  - Cleanup



JSP syntax

- **Declaration Tag** :-It is used to declare variables.
- **Java Scriptlets**:- It allows us to add any number of JAVA code, variables, and expressions.
- **JSP Expression**:- It evaluates and converts the expression to a string.
- **JAVA Comments** :- It contains the text that is added for information which has to be ignored.

Syntax: <%! Dec var %>

Example: <%! int var=10; %>

Syntax: <% java code %>

Syntax: <%= expression %>

Example:<%= num1+num2 %>

Syntax:<% -- JSP Comments %>

JSP Directives

- JSP directives are the elements of a JSP source code that guide the web container on how to translate the JSP page into its respective servlet. Syntax : <%@ directive attribute = "value"%>

- Directives can have a number of attributes that you can list down as **key-value pairs** . The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

Types of JSP directives

1. **Page Directives :** JSP page directive is used to define the properties applying the JSP page, such as the size of the allocated buffer, imported packages and classes/interfaces, tracking session is enabled or not, defining what type of page it is etc.

Example: <%@page contentType="text/html" pageEncoding="UTF-8"%>

```
<%@page import = "java.util.Date"%>
<%@page errorPage = "error.jsp" %>


<%Date d = new Date();%>
```

Types of JSP directives

2. **Include directive :** The include directive is used to include a file into a JSP page at the time of translation. These files can be html files, other jsp files etc. The advantage of using an include directive is that it allows code re-usability.
   The syntax of an include directive is as follows:

<%@include file = "file location"%>

Example: <%@ include file="header.jsp" %>

Types of JSP directives

3. **Taglib directive**: The taglib directive is used to define and declare custom tag libraries that are used in the JSP page. It's major application is JSTL(JSP standard tag library). It specifies the location of the tag library descriptor file and assigns a prefix that is used to identify the tags from that library.

Syntax: <%@ taglib uri="taglibURI" prefix="tagPrefix" %>

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

JSP Standard Tag Library Example

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%!  String[] names = {"Ruth", "Matilda", "Millicent", "Micah"}; %>
<c:forEach var="person" items="<%= names %>">
<c:out value="${person}" />
</c:forEach>
```

In the above code, we've used to taglib directive to point to the JSTL library which is a set of some custom-defined tags in JSP that can be used in place of the scriptlet tag (<%..%>).

Here, the prefix c is used in the <c:out> tag to tell the container that this tag belongs to the library mentioned above.

JSP Assignment #1

1. Perform all 23 exercises found on https://www.programmingempire.com/jsp-practice-exercise/
2. Using JSP, Design a Student admission App as per instructions below:
   i. Create 3 pages **Signup, Sign In** and **Admission**
   ii. SignUp **page attributes** (email, and password). Once signed up the user will be redirected to the login page.
   iii. The user must be logged in to have access to the student admission form (Apply authentication filters and Session management).
   iv. To the Student admission Page, add upload actions: Student passport picture. *Supported types*: .jpg or .png; and Certificates (Diploma). *Supported type*: only PDF format
   v. Using **JavaMail API** send submission confirmation email to the candidate, once the admission request is submitted.
   vi. Internationalize the application: 3 languages: **English, French and any Local language**

JSP - Actions

- **JSP actions** are special constructs that are used to perform specific tasks in JSP pages. There are several types of JSP actions, including:

  1. jsp:include: This action is used to include the contents of another file in the JSP page. It is similar to the include directive in servlets.
  2. jsp:forward: This action is used to forward the request to another resource, such as another JSP page or a servlet. It is similar to the forward method in servlets.

JSP - Actions

3. jsp:useBean: This action is used to instantiate a JavaBean and store it in a variable for later use. It can also be used to retrieve an existing bean.
4. jsp:setProperty and jsp:getProperty: These actions are used to set and get the properties of a JavaBean.
5. jsp:plugin: This action is used to embed a Java applet or other plugin in the JSP page.
6. jsp:param: This action is used to pass parameters to another resource, such as a servlet or JSP page.
7. jsp:scriptlet: This action is used to embed Java code directly in the JSP page.

jsp:include

Assume you have two JSP files:

"header.jsp" and "footer.jsp"

that contain the header and footer of the website, respectively:

```
<!DOCTYPE html>
<html>
<head>
    <title>My Website</title>
</head>
<body>
    <jsp:include page="header.jsp" />

    <!-- main content of the page goes here -->

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

jsp:forward

Here's an example of how to use the jsp:forward action in a JSP page:

Assume you have a JSP page called "login.jsp" that handles user login. After the user successfully logs in, you want to forward them to a different page called "dashboard.jsp". You can use the following code in your "login.jsp" page to achieve this:

```
<%
// perform login logic here...
if (loginSuccessful) {
    // forward to dashboard.jsp
    request.getRequestDispatcher("dashboard.jsp").forward(request, response);
}
%>
```

jsp:useBean

Assume you have a JavaBean called "Person" that represents a person's name and age, with the following properties and methods:

```
<jsp:useBean id="person" class="com.example.Person" />

Name: <jsp:getProperty name="person" property="name" /><br />
Age: <jsp:getProperty name="person" property="age" /><br />

<jsp:setProperty name="person" property="name" value="John Doe" />
<jsp:setProperty name="person" property="age" value="30" />
```

```
public class Person {
    private String name;
    private int age;

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

In this example, the jsp:useBean action is used to instantiate a new instance of the "Person" class and stores it in a variable named "person". The "id" attribute specifies the name of the variable, and the "class" attribute specifies the fully-qualified name of the JavaBean class.
The jsp:getProperty action is used to retrieve the "name" and "age" properties of the "person" object and display them on the page.
The jsp:setProperty action is used to set the "name" and "age" properties of the "person" object to new values.

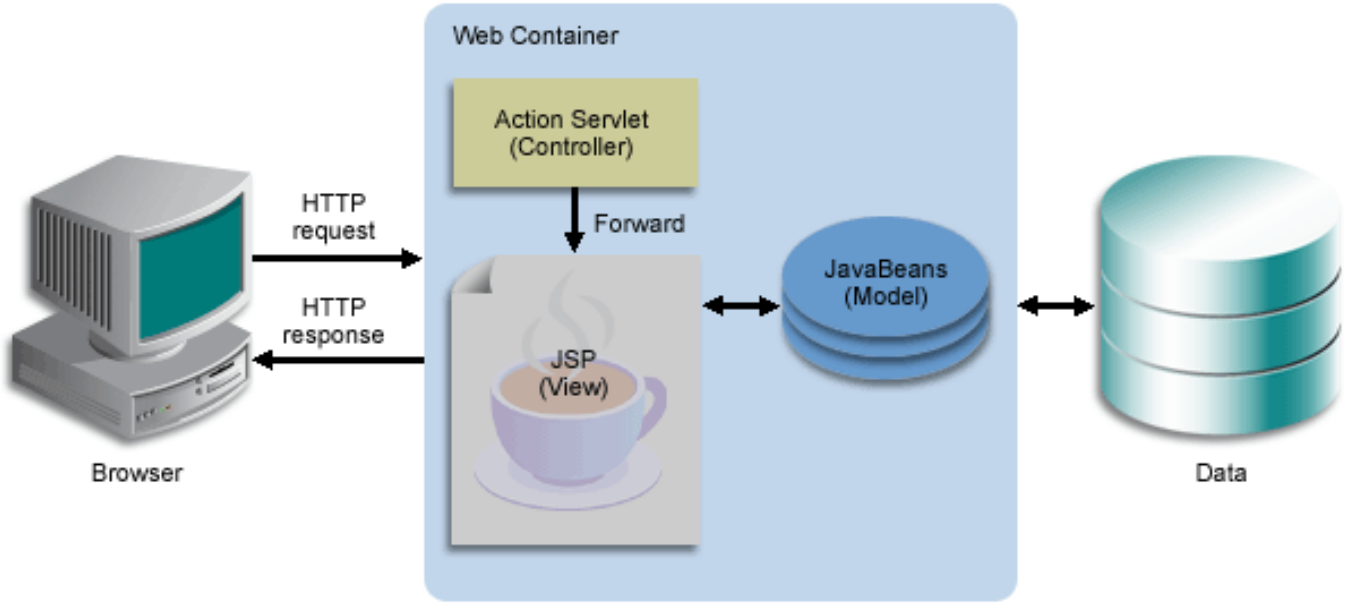
jsp:param

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
  <head>
    <title>My JSP Page</title>
  </head>
  <body>
    <h1>Hello, <%= request.getParameter("name") %>!</h1>
  </body>
</html>
```

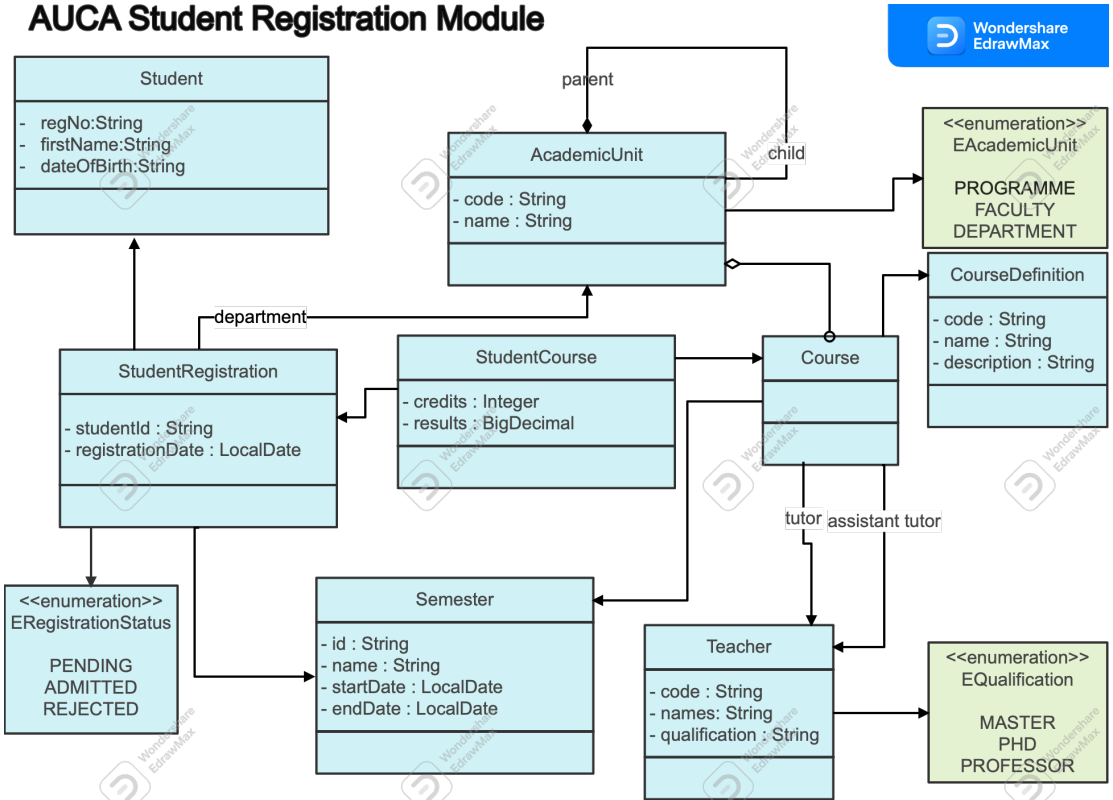
```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
  <head>
    <title>My JSP Page with Parameter</title>
  </head>
  <body>
    <%-- Set the parameter value --%>
    <jsp:param name="name" value="John Doe" />

    <%-- Include the first JSP page with the parameter --%>
    <jsp:include page="/myJspPage.jsp" />
  </body>
</html>
```

The <jsp:param> tag is used to pass parameters to a JSP page or a custom tag. Here's an example of using the <jsp:param> tag in a JSP page:

MVC Architecture in JSP



JSP Project: Implement the Student Registration Module



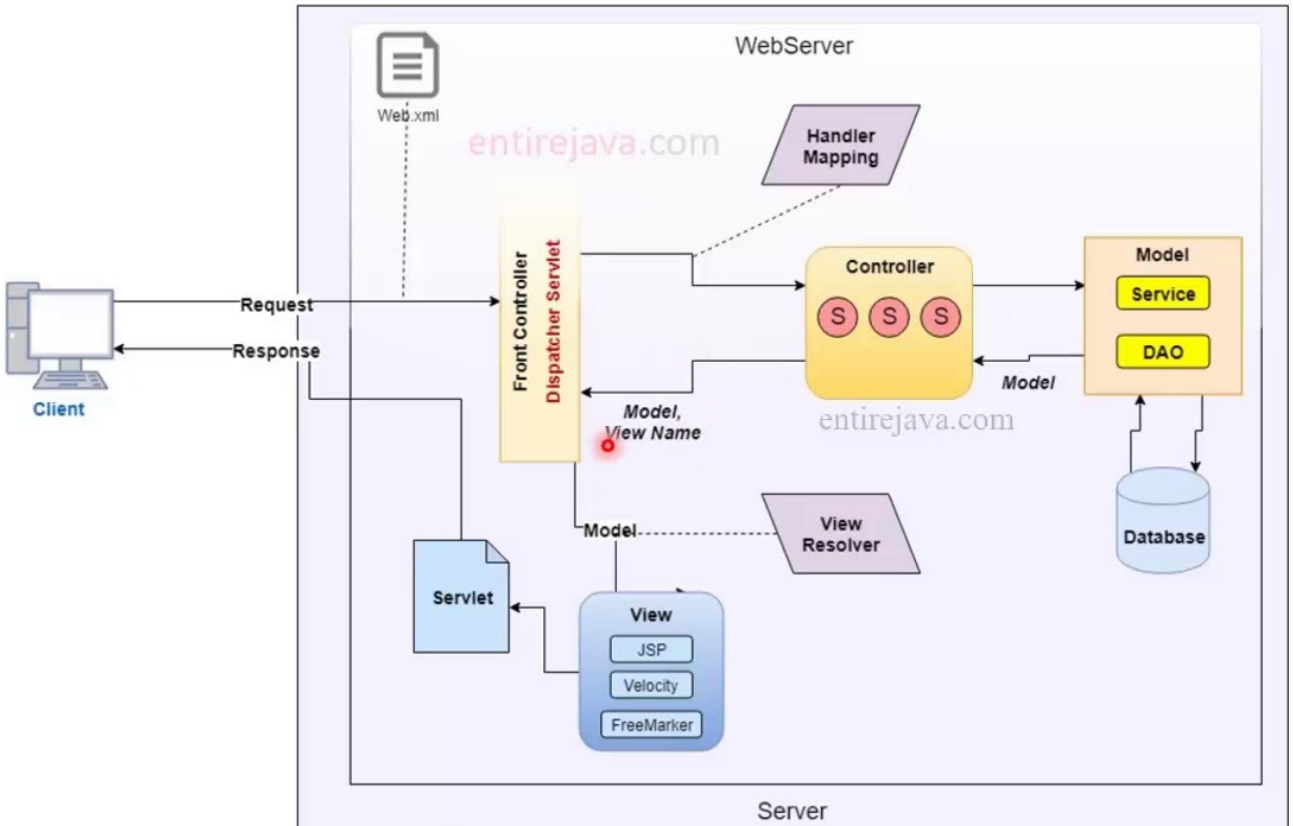
The following Lists will be generated:

a. List of Students per semester
b. List of Students per department and Semester
c. List of Students per course and Semester
d. List of Courses per Department and Semester
e. List of Courses per Student

NB: The tutor and assistant must apear on the list of courses

Spring MVC Framework

Spring MVC Architecture



Spring MVC Architecture