Module 3- Web Technologies in Java

❖ Introduction to HTML and its structure.

- HTML (HyperText Markup Language) is the standard language used to create and design web pages.
- It tells the browser what to display (text, images, links, forms, etc.).
- HTML is not a programming language; it’s a markup language that uses tags to structure content.
- Every web page you see online is built with HTML as its foundation.
```
<!DOCTYPE html>
        <head>
                <title>Page Title</title>
        </head>
        <body>
                <h1>Heading</h1>
                <p>Paragraph text</p>
        </body>
</html>
```

❖ Explanation of key tags

1. Anchor tag (Hyperlinks)
   Used to create links to other pages or sections.
   <a href="https://example.com">Visit Example</a>

2. <form> → Form tag (User input)
   Used to collect user input.
   ```
     <form action="submit" method="post">
             <label>Name: </label>
             <input type="text" name="username">
             <button type="submit">Submit</button>
    </form>
   ```

3. <table> → Table tag (Data representation)
   Displays data in rows & columns.
   ```
   <table border="1">
    <tr> <th>Name</th></tr>
   </table>
   ```

4. <img> → Image tag
   Embeds images in a webpage.
   <img src="photo.jpg" alt="My Photo" width="200">

5. List tags: <ul>, <ol>, <li>
   <ul> → Unordered list (bullets).
   <ol> → Ordered list (numbers).
   <li> → List item.
   ```
   <ul>
    <li>Apple</li>
   </ul><ol>
    <li>First</li> </ol>
   ```

6. <p> → Paragraph
   Defines a paragraph of text.
   <p>This is a paragraph of text.</p>

7. <br> → Line break
   Breaks text into a new line.
   Hello<br>World

8. <label> → Label for form inputs
   Describes input fields in forms.
   <label for="email">Email:</label>

❖ Overview of CSS and its importance in web design.

- ➢ CSS stands for Cascading Style Sheets.
- ➢ It is a stylesheet language used to describe the look and formatting of a document written in HTML.
- ➢ While HTML provides the structure (content and layout) of a webpage,
- ➢ CSS controls its appearance (colors, fonts, spacing, positioning, etc.).

1. Makes websites attractive
   Adds color, layout, and design.

2. Separates design from content
   HTML is for structure, CSS is for style.

3. Responsive design
   Makes sites look good on phones, tablets, and computers.

4. Easy to update
   One CSS file can style many pages.

5. Improves user experience
   Clean, professional, and consistent look.

❖ Types of CSS

1. **Inline CSS**
   Written directly inside an HTML element using the style attribute.
   Example:
   ```
   <h1 style="color: blue;">Hello World</h1>
   ```

2. **Internal CSS**
   Written inside a <style> tag in the head section of the HTML page.
   Example:
   ```
   <style>
    p { color: green; }
   </style>
   ```

3. **External CSS**
   Written in a separate .css file and linked to the HTML page. Example:
   ```
   <link rel="stylesheet" href="style.css">
   ```

❖ Definition and difference between margin and padding.

- Margin: Space outside an element — it creates distance between the element and other elements.
- Padding: Space inside an element — it creates distance between the element's content and its border.
- Example:
  ```
  div {
   margin: 20px; /* space outside */
   padding: 10px; /* space inside */
  }
  ```

❖ Introduction to CSS pseudo-classes like hover, focus, active, etc.

- CSS pseudo-classes are used to define the special state of an element. Common Pseudo-classes:
- hover – Applies a style when the user moves the mouse over an element.
- focus – Applies a style when an element (like an input box) is selected.
- active – Applies a style when an element is being clicked.
- Example:
  hover { color: red; }
  button:active { background-color: green; }
  input:focus { border-color: blue; }

❖ Use of pseudo-classes to style elements based on their state.

- Use of Pseudo-classes to Style Elements Based on Their State.
- Pseudo-classes are used to change the style of elements depending on their state or user interaction. Install it by following the installation wizard.
- They make web pages more interactive and user-friendly.

❖ Difference between id and class in CSS.

1. ID

- **ID** must be unique in a page (used for one specific element).
- **ID** is selected using a # (e.g., #header).
- **ID** is used for unique elements like navigation bar, footer, or main container.
- ID is often used to target a specific element with JavaScript.

2. Class

- Class can be used on multiple elements.
- **Class** is selected using a **.** (e.g., .title).

- **Class** is used for repeated styling, like buttons, headings, or cards.
- **Class** is used for selecting groups of elements.

❖ Usage scenarios for id (unique) and class (reusable)

- ID Used for unique elements that appear only once on a page
- Example:Header, Footer, or Main Section.

```
<div id="header"></div>
header { background-color: blue; }
```

- Class (.) Used for reusable styles that apply to multiple elements.
- Example: Buttons, Cards, or Text Blocks.

```
<div class="button"></div>
<div class="button"></div>
.button { background-color: green; }
```

❖ Overview of client-server architecture.

- Client-Server Architecture is a model where tasks and services are divided between:
- Client: The user-side device or application that requests services or resources.
- Server: The central system that provides services, processes requests, and stores data.
- How It Works
  1. The client sends a request to the server.
  2. The server processes the request.
  3. The server sends back a response to the client.

❖ Difference between client-side and server-side processing.

1) Client-side

- **Client-Side:** Processing occurs on the user’s device (browser).
- Usually faster because it doesn’t require communication with the server for every action.
- Less secure; users can view and modify the code.

- Form validation, animations, page updates (without reloading).
- Reduces server load and can improve user experience.

2) Server-side

- Processing happens on the web server.
- Can be slower due to server communication and data processing.
- More secure; code and data remain on the server.
- Database operations, authentication, data storage, business logic.
- Generates dynamic content before sending it to the browser.

❖ Roles of a client, server, and communication protocols.

1. Client

- Requests services or resources from the server.
- Examples: Web browser, mobile app.

2. Server

- Provides services, processes client requests, and sends back responses.
- Examples: Web server, database server.

3. Communication Protocols

- Rules that define how clients and servers exchange data.
- Examples: HTTP/HTTPS for web, FTP for file transfer, SMTP for email.

❖ Introduction to the HTTP protocol and its role in web communication.

- HTTP (Hyper Text Transfer Protocol) is the standard protocol used for communication between a web client (browser) and a web server.
- Role in Web Communication

1) Client sends a request – e.g., asking for a web page.
2) Server processes the request – retrieves the page or resource.
3) Server sends a response – page, data, or error message back to the client.

❖ Explanation of HTTP request and response headers

- HTTP Headers are key-value pairs sent between the client and server to provide additional information about the request or response.
- Request Headers :
  Sent by client to server, giving info about the request.
- Examples: Host, User-Agent, Accept
- Response Headers :
  Sent by server to client, giving info about the response.
- Examples: Content-Type, Content-Length, Server

❖ Introduction to J2EE and its multi-tier architecture.

- It supports multi-tier architecture, which separates an application into layers for better scalability and maintenance.
- Multi-tier Architecture Layers

1. **Client Tier (Presentation Layer)**
   - User interface, interacts with users.
   - Examples: Web browser, desktop apps.

2. **Web Tier (Presentation Logic)**
   - Handles client requests and responses.
   - Examples: Servlets, JSPs.

3. **Business Tier (Business Logic Layer)**
   - Processes business rules and logic.
   - Examples: EJBs (Enterprise Java Beans), Java classes.

4. **Enterprise Information System Tier (Data Layer)**
   - Stores and manages data
   - Examples: Databases, legacy systems.

❖ Role of web containers, application servers, and database servers.

1) **Web Container:**
   - Manages web components like Servlets and JSPs.
   - Handles HTTP requests and responses.
   - Example: Tomcat, Jetty

2) **Application Server:**
   - Manages business logic components like EJBs.
   - Provides services like transactions, security, and messaging.
   - Example: JBoss, WebLogic.

3) **Database Server:**
   - Stores and manages application data.
   - Handles queries, updates, and data integrity.
   - Example: MySQL, Oracle, SQL Server.

❖ Introduction to CGI (Common Gateway Interface).

- ➢ CGI is a standard protocol that allows web servers to interact with external programs or scripts to generate dynamic content.
- ➢ It enables web pages to respond to user input.
- ➢ CGI programs can be written in languages like Perl, Python, C, or Java.

1) Client sends a request to the web server.
2) Server executes the CGI program.
3) CGI program processes data and sends dynamic response back to the client.

❖ Process, advantages, and disadvantages of CGI programming.

➢ **Process of CGI Programming.**

- Client Request: User submits data via a form or URL.
- Server Executes CGI Program: Web server runs the external script/program

- Processing: CGI program processes input and generates output.
- Server Response: Output (HTML or data) is sent back to the client browser.

➢ **Advantages of CGI**.

- Simple and widely supported.
- Can be written in many programming languages (Perl, Python, C, etc.).
- Allows creation of dynamic web pages.

➢ **Disadvantages of CGI.**

- Slower performance for high traffic (new process created per request).
- Consumes more server resources.
- Harder to maintain and scale for large applications.

❖ Introduction to servlets and how they work.

➢ Servlets are Java programs that run on a web server to handle client requests and generate dynamic web content.

➢ They are part of Java EE and replace older technologies like CGI.

➢ Commonly used to create web applications and interact with databases.

1) Client Request: Browser sends a request (HTTP) to the server.
2) Servlet Container: Server's web container loads and executes the servlet.
3) Processing: Servlet processes the request (business logic, database access).
4) Response: Servlet sends dynamic content (HTML, JSON, etc.) back to the client.

❖ Advantages and disadvantages compared to other web technologies.

➢ Advantages of Servlets.

- Fast and Efficient: Runs in server memory, unlike CGI which creates a new process per request.
- Platform Independent: Written in Java, works on any OS with a Java-enabled server.
- Secure: Supports Java security features.
- Robust: Strong exception handling and multithreading support.
- Integration: Easily interacts with databases and other Java technologies.

➢ Disadvantages of Servlets.

- Complex for Beginners: Requires knowledge of Java and web concepts.
- No Direct HTML: HTML must be generated via code or JSP, which can be tedious.
- Server Dependency: Needs a servlet container or Java EE server to run.

❖ History of servlet versions

➢ Servlet 2.3 (2001)

- Introduced filters, listeners, and improved session management.

➢ Servlet 2.4 (2003)

- Enhanced error handling and deployment descriptors.

➢ Servlet 2.5 (2005)

- Compatible with Java EE 5, annotations introduced.

➢ Servlet 3.0 (2009)

- Annotations, asynchronous processing, and pluggability added.

➢ Servlet 3.1 (2013)

- Non-blocking I/O for better scalability.

➢ Servlet 4.0 (2017)

- Supports HTTP/2, improving performance for modern web apps.

❖ Types of servlets: Generic and HTTP servlets:

1) Generic Servlet

- ➢ Superclass: javax.servlet.GenericServlet.
- ➢ Protocol-independent (can work with any protocol, not just HTTP).
- ➢ Abstract class: You must override the service() method.
- ➢ Less commonly used for web applications.

2) HTTP Servlet

- ➢ Superclass: javax.servlet.http.HttpServlet.
- ➢ Protocol-specific: Designed for HTTP requests.
- ➢ Provides methods like doGet(), doPost(), doPut(), doDelete().
- ➢ Most widely used for web applications.

❖ Detailed comparison between HttpServlet and GenericServlet

1. GenericServlet

- ➢ Designed to handle **protocol-independent** requests (not tied to HTTP).
- ➢ can be used with any protocol (e.g., FTP, SMTP), though rarely used that way.
- ➢ No predefined methods for GET or POST.
- ➢ Suitable where only generic request/response handling is needed.
- ➢ No built-in session support.
- ➢ Rarely used in modern applications.

2. HttpServlet:

- ➢ Designed specifically for **handling HTTP requests** such as GET, POST, PUT, DELETE.
- ➢ Works **only with HTTP** protocol.
- ➢ Provides specialized methods such as: doGet(),doPost(),doPut(),doDelete(),doHead().
- ➢ Used commonly in MVC frameworks and REST applications.
- ➢ Full support for HTTP sessions via HttpSession.
- ➢ Highly used; foundation of many Java web frameworks and servlets.

❖ Explanation of the servlet life cycle: init(), service(), and destroy() methods.

1. Init()

- ➢ Called once when the servlet is first loaded.
- ➢ Used to initialize resources (like database connections).

2. Service()

- ➢ Called for every client request.
- ➢ Handles request processing and generates a response.
- ➢ In HttpServlet, this method internally calls doGet(), doPost(), etc.

3. Destroy()

- ➢ Called once before the servlet is removed from memory.
- ➢ Used to release resources (like closing database connections).

❖ How to create servlets and configure them using web.xml.

1) Create Servlet Class

```
public class HelloServlet extends HttpServlet {
    protected void doGet (HttpServletRequest req,
    HttpServletResponse res)
        throws IOException {
        res.getWriter().println("Hello, World!");
        }
      }
```

2) Configure in web.xml

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

❖ Explanation of logical URLs and their use in servlets.

- Logical URLs are friendly URLs that are mapped to servlets via web.xml or annotations, rather than using the physical file path.
- They hide the actual servlet class location from the user.
- Make URLs clean, readable, and easier to manage.
- Example:

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

❖ Overview of ServletConfig and its methods

- ServletConfig is an interface provided by the servlet API that allows a servlet to access its configuration information from web.xml.
- Each servlet has its own ServletConfig object.
- Used to read initialization parameters and get servlet context.
- **Methods:**
  - **getInitParameter(String name):** Retrieves the value of a servlet's initialization parameter.
  - **getInitParameterNames():**Returns all initialization parameter names as an Enumeration.
  - **getServletContext():**Returns the ServletContext object for communication with the server.
  - **getServletName():**Returns the name of the servlet.

❖ Explanation of RequestDispatcher and the forward() and include() methods.

- RequestDispatcher is an interface used to forward a request from one servlet/JSP to another, or include content from another resource.

- Helps in modularizing web applications.
- Objects are obtained using request.getRequestDispatcher("path").

1. **forward(request, response):**
    - Forwards the request to another servlet or JSP.
    - Client does not see the new URL.
    - Stops execution of the current servlet after forwarding.
2. **include(request, response):**
    - Includes content from another servlet or JSP in the same response.
    - Execution of current servlet continues after including.

- **Example:**

```
RequestDispatcher rd =
request.getRequestDispatcher("next.jsp");

rd.forward(request, response);

rd.include(request, response);
```

❖ Introduction to ServletContext and its scope

- ServletContext is an interface that allows communication between a servlet and the web server.
- It provides information about the web application and resources.
- Shared by all servlets in the same web application.
- Application-wide scope:
    - Data stored in ServletContext is accessible by all servlets and JSPs in the web application.
    - Useful for sharing resources, configuration, or data across the application.

❖ How to use web application listeners for lifecycle events

- Listeners are special classes in Java EE that monitor and respond to events in a web application, such as servlet context changes, session creation, or attribute modifications.
- They implement listener interfaces provided by the Servlet API.

- Registered in web.xml or using annotations.
- Common Listener Types:
  - **ServletContextListener:** Tracks web application start and stop.
  - **HttpSessionListener:** Tracks session creation and destruction.
  - **ServletRequestListener:** Tracks request initialization and destruction.