P Latha, R Praghatha, T Vinotha, S. Kanimozhi, K.Sudha

Fundamentals of Web Technology

DeepScience

Fundamentals of Web Technology

P Latha

Department of Computer Science and Engineering at St. Jospeh's College of Engineering and Technology- Thanjavur

R Praghatha

Department of Computer Science and Engineering at St. Jospeh's College of Engineering and Technology- Thanjavur

T Vinotha

Department of Computer Science and Engineering at St. Jospeh's College of Engineering and Technology- Thanjavur

S. Kanimozhi

Department of Computer Science and Engineering at St. Jospeh's College of Engineering and Technology- Thanjavur

K.Sudha

Department of Computer Science and Engineering at St. Jospeh's College of Engineering and Technology- Thanjavur

DeepScience

Published, marketed, and distributed by:

Deep Science Publishing, 2025
USA | UK | India | Turkey
Reg. No. MH-33-0523625
www.deepscienceresearch.com
editor@deepscienceresearch.com
WhatsApp: +91 7977171947

ISBN: 978-93-7185-752-9

E-ISBN: 978-93-7185-630-0

https://doi.org/10.70593/978-93-7185-630-0

Citation: Latha, P., Praghatha, R., Vinotha T., Kanimozhi, S., & Sudha, K. (2025). *Fundamentals of Web Technology*. Deep Science Publishing. https://doi.org/10.70593/978-93-7185-630-0

COURSE OUTLINE

CHAPTER 1

First Chapter is about “*Principles of Web Systems and Network Interaction*” this chapter is contributed by the author Mrs P Latha, Assistant Professor in the Department of Computer Science and Engineering at St. Jospeh’s College of Engineering and Technology- Thanjavur.

CHAPTER 2

Second Chapter is about “*Foundation of Markup Languages*” this chapter is contributed by the author Mrs R Praghatha, Assistant Professor in the Department of Computer Science and Engineering at St. Jospeh’s College of Engineering and Technology- Thanjavur.

CHAPTER 3

Third Chapter is about “*Web Page Styling*” this chapter is contributed by the author Mrs T Vinotha, Assistant Professor in the Department of Computer Science and Engineering at St. Jospeh’s College of Engineering and Technology- Thanjavur.

CHAPTER 4

Fourth Chapter is about “XHTML” this chapter is contributed by the author Mrs S. Kanimozhi Assistant Professor in the Department of Computer Science and Engineering at St. Jospeh’s College of Engineering and Technology- Thanjavur.

CHAPTER 5

Fifth Chapter is about “SERVER-SIDE PROGRAMMING JAVA SERVLETS” this chapter is contributed by the author Mrs. K. Sudha, Assistant Professor in the Department of Computer Science and Engineering at St. Jospeh’s College of Engineering and Technology- Thanjavur.

TABLE OF CONTENT

Chapter 1: Principles of Web Systems and Network Interaction

1.1 Introduction to Global Network

The Internet is a global network of millions of computers and devices that communicate with one another. It works by connecting smaller networks together, allowing people to share information, access websites, send messages, and use online services.

At its core, the Internet relies on three essential ideas:

1. **Networks and Connectivity**

    - A network refers to a collection of inter-related devices that have the ability of sharing information.
    - The Internet consists of a big pool of networks that are interconnected or commonly referred to as a network of networks.
    - The communication is conducted through cables, Wi-Fi, satellites among others.

2. IP Addresses

- Every Internet connected device has an IP address of its own.
- This address functions as a home address, it informs the network with regards to where the data should be sent..

3. **Protocols**

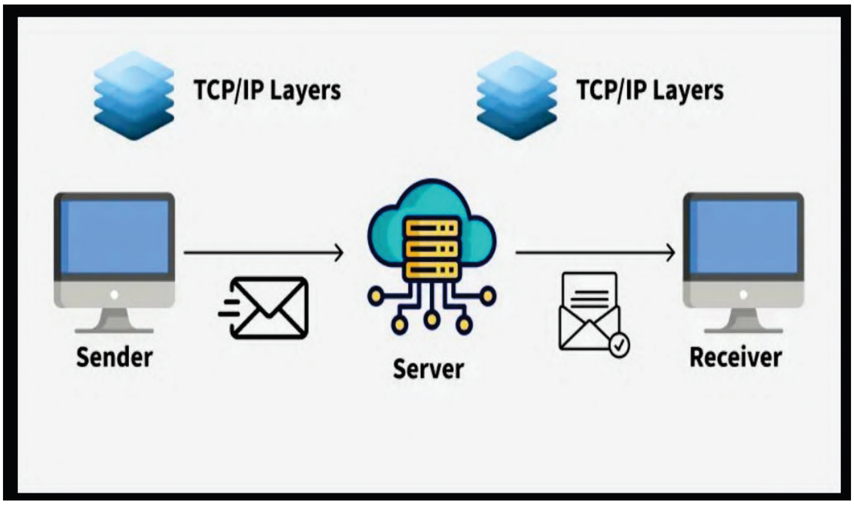
- The Internet works because all devices follow shared rules called protocols.
- The most important are:
    - TCP/IP – for sending and receiving data reliably.
    - HTTP/HTTPS – for loading web pages.
    - DNS – for translating domain names (like *example.com*) into IP addresses.

1.2 Basic Internet Protocols

Internet protocols are standardized communication rules that specify how data is formatted, routed, transmitted, and validated across interconnected networks. Core examples include IP for packet addressing, TCP for dependable transport, and HTTP for web-level interactions.

1.2.1 TCP/IP

The Transmission Control Protocol /Internet Protocol (TCP/IP) forms the basis of the architecture of protocols of the contemporary network. It regulates the process of packetization, addressing, transmission, routing, and reception of data whereby TCP can be relied upon to provide reliability and order delivery whereas IP can be used to provide end-to-end addressing and routing.



1.2.2 UDP, DNS, and Domain Names

UDP

UDP is a connectionless network protocol that is thin and therefore transmits data fast without delivering guarantees and can therefore be used in real time applications like streaming, games and voice calls where low latency is more important than reliability.

UDP is a fast transport protocols used to send packets without connecting or treating them to arrive in an intact form hence is applicable in applications that value fast delivery over accuracy.

DNS

Domain Name System(DNS) is used to convert the names that people read into the IP addresses. Majority of DNS requests are sent using UDP protocol which is lightweight and connectionless protocol of transport which serves to communicate using light and compact packets tightly.

DNS, or Domain name service, is a distributed naming service, which encodes human legible domain names into numeric IP addresses, so that via the use of DNS, users are spared the hassle of memorizing the complex numbers, in order to access websites and network services.

DNS is a hierarchical address system, when domain names which are easy to memorize are transformed into IP address used by computers to allow easy navigation to websites and online resources.

Domain Names

A domain name is a human-friendly, unique address on the internet, like google.com, that translates to a complex numerical IP address. It is used to access websites, email services, and other online resources, making it easier for people to remember and navigate the web. You can purchase and register a domain name from a registrar, and it functions as your online identity and a consistent web address, even if the underlying server location changes. How domain names work Human-readable address: A domain name is a memorable string of text that points to a specific website.

IP address: Every device connected to the internet has a unique, numerical IP address (e.g.,). Domain Name System (DNS): DNS is a system that translates the domain name you type into a browser (like example.com) into its corresponding IP address so your computer can locate the website. Analogy: Think of the domain name as the street address of a building, and the IP address as the building's physical coordinates. The DNS is the map that connects the two. Why use a domain name Memorability: It's much easier to remember google.com than a string of numbers like 172.217.160.142. Branding and credibility: A unique domain name establishes your brand identity online and builds credibility with customers. Flexibility: You can change your website's hosting server and IP address without affecting your domain name, as long as you update the DNS records to point to the new location.

Important elements of a domain name Top-Level Domain (TLD): The final element of the domain name i.e. .com, .org or .net.. A subdomain is an extension of a parent domain used to organize content, for example mail.google.com or docs.google.com. Domain Name: The unique name chosen by the user, such as google in google.com. Introduction to domain names - Google workspace Learn Overview A domain name (also commonly known as a domain name) is a name which is easy to remember and which is related to a physical Internet address stemmed out of an IP address.

Higher-level protocols are sets of rules that control the logical part of data communication, managing tasks like establishing connections, handling errors, and ensuring data integrity beyond basic physical transmission. They give the architecture; they stipulate the understanding of how information is translated and shared including the HTTP protocol in web browsing, SMTP in email or protocols in managing resources in a distributed system.

These protocols are built over physical and data link layer protocols which are the lower-level protocols. Another important role of the higher-level protocols Connection management Initiate, create, and end data connections with other devices. Data integrity: Carry out checking and fixing of the data to ascertain that the received data is correct.. Logical addressing: Provide a way to identify and address devices on the network. Control: Manage and control the flow of data transmission. Standardization: Define data types, parameters, and common rules to allow devices from different manufacturers to communicate effectively. Examples Higher level protocols HTTP (Hypertext Transfer Protocol): This is the one used to transfer web pages and other things on the World Wide Web. SMTP (Simple Mail transfer protocol): This is used in connection with sending e-mails. TCP/IP (Transmission Control Protocol/Internet Protocol): A set of protocols that is the basis of the Internet upon which TCP and IP protocols operate based on reliable data transmission and addressing and routing respectively.

FTP (File transfer protocol): It is applied in transfer of files among computers. CANopen A higher-layer protocol standard of CAN (Controller Area Network) buses, applied in industrial automation and robotics. Explained Low-level protocols and High-level protocols 23 Apr 2025 VbE Higher-level protocols. The logical part of data communication is controlled by high-level protocols. They provide the following... ComputerNetworkingNotes Higher Layer Protocols - Kvaser The CAN standard is the standard that specifies the hardware (the physical layer - there are many) and the communication on that bare minimum level. (“the data... Kvaser Web Technology: Theory and Practice [Book] - O'Reilly Protocols operate at multiple levels during communication between systems on the web as is seen in the OSI model. Key to using the... O'Reilly Media Show all

Higher-level protocols define how applications should structure, interpret, and reliably exchange data, building on lower-layer services to manage connections, ensure accuracy, and coordinate communication between networked systems.

Higher-level protocols provide the rules that guide how software systems communicate, overseeing tasks like connection setup, error handling, and data formatting so applications can exchange information in a consistent and reliable way.

1.3 The World Wide Web

The World Wide Web (WWW) is a system of interconnected public webpages accessible through the internet, allowing for the easy sharing and retrieval of information. It is not the same as the internet, but is one of many applications that run on it. The Web is made up of components like hyperlinks, the HTTP protocol, and Uniform Resource Locators (URLs), and enables multimedia content such as text, images, audio, and video.

Web structures information in an organized manner where one can easily navigate through web pages and facilitating the interactive aspect of the web like forms, multimedia, and dynamic contents.

It is also based on web standards and technologies such as HTML, CSS and JavaScript such that it presents itself and works within various browsers and devices with relative consistency.

Web provides a global network of available information that can be searched, shared and updated in real-time by connecting the content with URLs and links.

1.3.1Hypertext Transport Protocol

One of the protocols employed in the Internet is the Hypertext Transport protocol.

The collection of regulations that allow the web browsers and servers to interact, to provide and obtain web pages, images, videos and other web examples is known as HTTP (Hypertext Transfer Protocol).

Based on client-server architecture, HTTP has a request-response framework, in which a browser submits a request of a resource to the server, which responds to the request with an equivalent response data.

Being an application-layer protocol, HTTP will enable the exchange of hypertext documents over the internet but it is stateless, by which is meant every request needs no integration with a prior request.

The World Wide Web has its foundation in HTTP and utilizes it to ensure the content stored on the servers can be accessed, relayed, and viewed easily by the client computing equipment.

Upon entering the URL in a browser,HTTP is involved in the process of communication between the browser and the web server, including the process of requesting, getting responses and rendering the contents.

1.4 HTTP Request Message

An HTTP request message is the information that is sent by a client (typically a web browser) to a server to demand a particular resource, i.e. a web page, image or file.. It follows a structured format that allows the server to understand what the client is asking for and how to respond.

Key components of an HTTP request message:

1. **Request Line:**
    - Specifies the HTTP method, the resource URL, and the HTTP version.
    - Example: GET /index.html HTTP/1.1
        - GET → method
        - /index.html → requested resource
        - HTTP/1.1 → version
2. **Headers:**
    - Provide additional information about the request or the client.
    - Examples include:
        - Host: the domain name of the server
        - User-Agent: the client software making the request
        - Accept: types of content the client can handle
3. **Blank Line:**
    - Separates the headers from the optional body.
4. **Body (Optional):**
    - Contains data sent to the server, usually with methods like POST or PUT.
    - For example, form data submitted by a user.

Example of a simple HTTP GET **request:**

GET /about.html HTTP/1.1

Host: www.example.com

User-Agent: Mozilla/5.0

Accept: text/html

1.4.1 Overall Structure

The overall structure of an HTTP request message can be broken down into four main parts:

1. Request Line
    - The first line of the request.
    - It specifies:
        - HTTP Method: Action the client wants to perform (e.g., GET, POST, PUT, DELETE).
        - Request-URI: The path of the resource being requested (e.g., /index.html).

        - HTTP Version: The version of the HTTP protocol used (e.g., HTTP/1.1).
    - Example:
    - GET /home.html HTTP/1.1
2. Request Headers
    - Key-value pairs that provide additional information about the request or client.
    - Common headers include:
        - Host: The domain name of the server
        - User-Agent: The client software
        - Accept: Types of content the client can process
        - Authorization: Credentials for authentication
    - Example:
    - Host: www.example.com
    - User-Agent: Mozilla/5.0
    - Accept: text/html
3. Blank Line
    - A single empty line separating the headers from the body.
    - Indicates the end of the headers section.
4. Message Body (Optional)
    - Contains data sent to the server, usually with methods like POST or PUT.
    - For example, form submissions, JSON payloads, or file uploads.
    - Example:
    - name=JohnDoe&email=john@example.com

1.4.2 HTTP Version

HTTP/0.9: The first version that was introduced in 1991 was a one-line protocol which only allowed transmission of simple HTML documents without any header by use of GET. HTTP/1.0: This version was introduced in 1996, which introduced the ability to be extended through the introduction of: headers status codes and new methods such as POST and HEAD. Nevertheless, with every request a new connection was needed. HTTP/1.1: This is the original version that was introduced in 1997 and that are in use to date. It added long-running connections to enable multidirected connections to be made within one TCP connection, enhanced caching, and pipelining. One disadvantage was head-of-line blocks when one slow request had the ability to incur delay on other requests. HTTP/2: This was introduced in 2015, and overcame the performance problems of the HTTP/1.1 by adopting a binary representation, and having the capability to multiplex requests and defaults that do not block each other. It also contained header compression which was used to minimize the overhead and server push which proactively pro-sends resources. HTTP/3: The most recent one standardized in 2022 based on QUIC transport protocol that applies UDP rather than TCP. This circumvents TCP-level head-of-line blocking, offers quicker connection handshakes and manages changes in the mobile networks with better ease.

1.4.3 Request-URI

A request-uri is an important element in the HTTP response as stipulated in the internet protocol (RFC 2616, Section 5.1.2). It can be used to determine the exact resource that the method (e.g. GET, POST) of the request should be applied.

Request-Uri is one of the most important components of an HTTP request that is stipulated in HTTP/1.1 (RFC 2616, Section 5.1.2) that identifies the resource in the case of which the request method (GET or POST) should be addressed.

The URL in a web request is a file or document (the resource) that the client commands the server to perform some action on, and acts as the object of such actions as in a GET, POST, or PUT request.

1.4.4 Request Method .

1. The HTTP request methods identify the kind of an operation that a client desires to have the server carry out, which may be retrieving, creating, updating, or deleting of a resource.
2. Such ways are common in HTTP requests: GET requests are used to retrieve information, POST requests are used to submit information, PUT requests are used to edit resources, and DELETE requests are used to delete resources.
3. These are techniques that enable servers in the webcommunication to know what a request is and how to reply accordingly, the core of a RESTful web communication.
4. Having identical actions by using the HTTP verbs, clients and servers communicate in a predictable manner which allows reliable handling of data exchanges between varied web applications.
5. The HTTP method has a certain semantics attached to it, not defining what the server should do with the request but determining whether the operation is idempotent or safe.
6. In addition to the standard verbs, HTTP also adds such methods as HEAD, OPTIONS, and Patch in order to provide custom interactions and control over the resources in fine-grained manners.

1.5 HTTP Response Message

An HTTP response message is the message in which a server sends to a client, in response to an HTTP request. It informs about the outcome of the request and in case it succeeds the requested resource.

An HTTP response message consists of three main parts:

- **Status Line:**

This is the first line of the response and contains:

- The HTTP version (e.g., HTTP/1.1).

  - A three-digit status code indicating the outcome of the request (e.g., 200 for success, 404 for Not Found, 500 for Internal Server Error).
  - A reason phrase, a short textual description of the status code (e.g., OK, Not Found, Internal Server Error).
- **Response Headers:**

These are optional lines following the status line, providing additional information about the response, the server, or the content being sent. Examples include:

  - Content-Type: Specifies the media type of the response body (e.g., text/html, application/json).
  - Content-Length: Indicates the size of the response body in bytes.
  - Server: Identifies the web server software.
  - Set-Cookie: Sends cookies to the client.
- **Response Body:**

This is the optional part of the response message that contains the actual data requested by the client. For example, if a client requests a web page, the response body would contain the HTML content of that page. The body is separated from the headers by a blank line. The presence and size of the body are indicated by the status line and HTTP headers.

1.5.1 Response Status Line

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (100 – 199)
2. Successful responses (200 – 299)
3. Redirection messages (300 – 399)
4. Client error responses (400 – 499)
5. Server error responses (500 – 599)

1.5.2 Response Header Fields

Response header fields are metadata sent from a server to a client that provides additional information about the response, such as its content type, caching instructions, or server details. They give context about the response itself, and common examples include Content-Type, Server, Date, and Cache-Control. Response headers can also be used to manage redirects (Location), handle authentication (WWW-Authenticate), or inform the client about server status (Retry-After).

Examples of response header fields

- **Content-Type**: Indicates the media type of the resource in the body, such as text/html or application/json.
- **Server**: Provides information about the server software that handled the request.

- **Date**: Shows the timestamp when the response was generated.
- **Cache-Control**: Gives directives for how the response can be cached.
- **Location**: Used in redirection responses to point to a new URL.
- **WWW-Authenticate**: Defines the authentication method needed to access the resource.
- **Retry-After**: Indicates when the user-agent can retry a request, often used with a 503 Service Unavailable status.
- **ETag**: Provides an entity tag for the requested variant, useful for caching and versioning.

1.5.3 Cache Control

Cache-Control is an HTTP header that gives web developers control over how browsers and intermediary caches store and serve website resources. It uses directives, such as max-age to set an expiration time, or no-store to prevent caching, to ensure content is delivered quickly while staying up to date when needed. This helps improve performance by reducing the need for users to re-download resources from the server.

1.6 Web Clients .

A web client is a software application or device that sends requests to a web server to access and display content from the internet. The most widespread ones are web browsers such as Chrome, Firefox, and Safari, however, other applications and mobile applications based on the Hypertext Transfer Protocol (HTTP) to interact with a server are also viewed as the examples of the term..

- A web client, such as a browser, acts as the front-end or user-side of the internet experience.
- When a user requests a webpage, the client sends an HTTP request to the appropriate web server.
- The web server responds by sending the requested data (e.g., HTML, images, files) back to the client.
- The web client then processes and renders this data, presenting it to the user in a readable format.

1.6.1 Basic Browser Functions

A simple web browser functionality is the ability to access web pages by making and receiving content via a server, browsing the web through the features such as tabs, history, and other features, rendering web pages, any code translation into a user-friendly visual representation. Users also can input such information as clicks and submissions using form, Browsers also offer security and run Web applications.

Core functions

Internet access: When a user wants to access the internet, he/she will enter an URL in the address bar and the browser will make a request to a web server to access the desired page or resource.

- Web page rendering Web page rendering is the process of translating webpage code (such as HTML, CSS, and JavaScript) into text, graphics, and other user-interactive elements on the display of the customer.
- Navigation: There exists tools to navigate the web in browsers such as back/forward buttons, a history log and the ability to open several pages in tabs.
- Execution of web applications: The more advanced browsers have the capability of managing complicated, dynamic programs such as email and video players like the web page itself.
- Utilizing user input: Browsers process the user input like clicks on hyperlinks, text typed into the forms or search queries.

1.6.2 URLs

A URL, also known as Uniform Resource Locator, is a unique address that is used to find out a resource on the internet and that could be a web page, picture or file. It acts as a street address of the web interface and it instructs a browser how and where to access the requested content.. URLs can be used to navigate websites by typing them into an address bar or by clicking a hyperlink.

- **Protocol:** Specifies the method of access, such as http (Hypertext Transfer Protocol) or https (Secure HTTP).
- **Domain name:** The name of the website, such as google.com.
- **Path:** The specific location of a file or page on the server, such as /products/ads-commerce/.
- **Optional elements:** URLs can also include a subdomain (support.google.com), a port number, or other parameters.

Key functions of URLs

- **Resource location:** URLs identify and locate a unique resource online.
- **Web navigation:** They enable users to access and navigate the web by providing addresses for websites and pages.
- **Information retrieval:** A URL provides the instructions for a browser to request and retrieve content from a web server.

1.6.3 User-Controllable Features

User-controllable features are elements in a digital interface that give users the ability to manage their experience, such as **undo/redo** options, **customizable settings**, and **intuitive navigation controls**. These features empower users to correct mistakes, personalize their interface, and explore options freely without feeling trapped.

Examples of user-controllable features

- **Undo and redo:** The ability to reverse or reapply recent actions. This is a crucial feature for correcting mistakes without fear of losing progress.
- **Customization:** Users can personalize the interface to their liking, such as changing themes, adjusting font sizes, or altering a layout.
- **Navigation and exit options:**

   - **Back links:** Allows users to return to a previous page or screen.
   - **Cancel or close buttons:** Lets users exit a process or view without completing it.
   - **Confirmation dialogs:** Informs users before a significant change occurs and asks for confirmation, giving them a chance to cancel.
- **Input controls:**
   - Dropdown menus, sliders, checkboxes, and radio buttons for making selections.
   - Text fields for data input.
   - Buttons to initiate specific actions.
- **Accessibility features:** Functions like resizing text, audio controls, or other options that allow users to adapt the product to their specific needs.

1.7 Web Servers

A web server is a combination of **hardware and software** that stores, processes, and delivers web content like web pages, images, and videos to users over the internet using the **HTTP protocol**. When a user's browser requests a resource, the web server responds by sending the requested content or an error message if the resource cannot be found.

- **Client request**: A user types a URL into their browser, which sends an HTTP request to the server where the website's files are stored.
- **Server response**: The web server software receives the request, finds the corresponding file, and sends it back to the user's browser via HTTP.
- **Content delivery**: The browser then assembles and displays the website for the user.
- **Error handling**: If the server cannot find the requested file, it sends an error message, such as a "404 Not Found" error.

1.7.1 Server Features

Server capabilities combine hardware elements such as fast CPUs, large memory, and high-performance storage with essential functions like reliability, remote administration, and easy scalability.

AI Overview

Remote management: Gives administrators the ability to use the server remotely.

Powerful hardware: Compres such components as alternative power supply, intensive cooling, mother boards that are server-specific.

SUMMARY:

Web Systems and Network Interaction describes the principle concepts of the way the Internet and the Web work. It has the fundamental protocols (TCP/IP, HTTP, DNS), Web (URLs, browsers, servers) and organizational framework of the HTTP communication. It as well presents the interaction between client and server, requesting and delivery of web pages, and the interaction among network protocols to provide connectivity of the world..

Chapter 2: Foundations of Markup Languages

2.1 Introduction to Markup Languages

Markup languages refer to systems of annotating text in a manner that facilitates understanding of the structure, formatting and meaning of the text by machines and human beings. They create labels to identify points in a document, which, in turn, allows that content and presentation can be separated.

One of the most popular markup languages is HTML, XML, and XHTML that are used with a varying purpose but a similar role.. Understanding their foundations is essential for designing structured, well-formatted, and interoperable web documents.

A markup language uses a system of **tags**, **elements**, and **attributes** to annotate information so that machines and humans can interpret content meaningfully. Over the years, markup languages have evolved from the complex SGML system to the more structured and web-friendly languages such as HTML, XML, and XHTML.

Understanding these three languages is not only essential for web developers but also for data engineers, application programmers, and researchers working with structured data.

2.2 Evolution of Markup Languages

2.2.1 From SGML to HTML

- SGML (Standard Generalized Markup Language) is the parent language from which HTML was derived.
- HTML (Hypertext Markup Language) was introduced to display and link documents through the World Wide Web.

  - Documents were plain text
  - Formatting was embedded in code
  - No clear distinction existed between content and presentation
  - Data sharing between systems was difficult
  - Software needed a standard mechanism to understand structure

2.2.2 Emergence of XML

XML (eXtensible Markup Language) was designed to address HTML's limitations in data representation by allowing users to define their own custom tags.

2.2.3 The Birth of XHTML

XHTML (Extensible Hypertext Markup Language) combines HTML’s functionality with XML’s strict syntax rules, enabling better document structure and compatibility.

Features of SGML

- Allows users to define custom markup languages
- Extremely flexible
- Supports large documentation systems
- Allows document type definitions (DTDs)
- Separates structure, semantics, and formatting

Limitations of SGML

- Very complex syntax
- Hard to implement
- Requires powerful processors
- Not suitable for beginner users
- Not ideal for web-based environments

Industries Using SGML

- Aerospace (aircraft manuals)
- Defense documentation
- Technical manuals
- Legal and government documents

2.3.3 HTML Tags: Types and Rules

Types of Tags

1. **Container tags**

   <html>, <body>, <div>, <p>

   → Need opening and closing tags.
2. **Empty tags**

   <br>, <img>, <hr>

   → No closing tag required.
3. **Inline vs Block Elements**
   - Inline: <span>, <a>, <strong>
   - Block: <div>, <p>, <section>
4. **Semantic Tags (HTML5)**

   <article>, <header>, <nav>, <footer>, <figure>

2.3.4 HTML Attributes

Attributes modify tag behavior:

- id
- class

- style
- src
- href
- alt
- width, height

Each attribute has a **name-value pair**.

2.3.5 Limitations of HTML

- Cannot describe data meaning
- Lacks extensibility
- Not self-descriptive
- Cannot enforce data validation
- Focuses on **presentation**, not structure

These limitations led to the creation of XML.

2.4 XML (eXtensible Markup Language)

XML is a **metadata-driven**, **self-descriptive**, **strict**, and **extensible** markup language designed for data storage and interchange.

2.4.1 Why XML?

XML was introduced for:

- Platform-independent data exchange
- Structured representation of information
- Custom tag creation
- Validation using DTD/XSD
- Machine-friendly parsing
- A bridge between heterogeneous systems

XML is now used in:

- Web services (SOAP, WSDL)
- Office file formats (docx, xlsx, odt)
- Databases (XML DB)
- Configuration files
- Mobile systems
- Scientific data exchange
- APIs and IoT systems

2.4.2 XML Syntax and Structure

An XML document contains:

1. XML Declaration
2. Root element
3. Child elements
4. Attributes
5. Text content
6. Comments
7. Optional DTD/XSD for validation

Example XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<student>
  <name>Praghatha</name>
  <department>CSE</department>
  <year>2025</year>
</student>
```

2.4.3 XML Rules

Well-formed XML Requirements

1. Must have a root element
2. Tags must be closed
3. Tags must be nested properly
4. Case-sensitive
5. Attribute values must be in quotes
6. No pre-defined tags — fully extensible

2.4.4 XML vs HTML

Feature	**HTML**	**XML**
Purpose	Display data	Store & structure data
Flexibility	Fixed tags	Custom tags
Validation	No strict validation	Strong validation
Syntax strictness	Lenient	Very strict
Use cases	Web pages	Data transport

2.4.5 DTD, XSD, Namespaces

XML supports validation using:

- **DTD (Document Type Definition)**
- **XSD (XML Schema Definition)**
- **Namespaces** to avoid naming conflicts

2.5 XHTML (Extensible HyperText Markup Language)

XHTML is a hybrid language that combines:

- **HTML semantics**
- **XML strictness**

It enforces well-formed syntax and is parsed using XML parsers.

2.5.1 Why XHTML?

HTML became too permissive, causing:

- Browser incompatibilities
- Broken syntax
- Difficult machine processing

XHTML solved these through stricter rules.

2.5.2 XHTML Document Structure

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>XHTML Example</title>
</head>
<body>
<p>Hello World</p>
</body>
</html>
```

2.5.3 XHTML Rules

1. All tags must be lowercase
2. All tags must be closed
3. Empty elements must be self-closed (<br />)
4. Attribute values must be inside quotes
5. Nested elements must be properly arranged

2.6 Key Comparisons Between HTML, XML, and XHTML

2.6.1 Structural Differences

Feature	**HTML**	**XML**	**XHTML**
Purpose	Display	Data	Web + XML
Strictness	Low	High	Very high
Tag Case	Not case sensitive	Case sensitive	Lowercase
Closing tags	Optional	Mandatory	Mandatory
Empty tags	Allowed	Not allowed	Must be self-closed

2.7 Integration and Interoperability

HTML, XML, and XHTML work together in several systems:

Examples

- SVG and MathML inside HTML5 pages
- XHTML documents displayed by modern browsers
- XML-based RSS feeds displayed in HTML
- SOAP/XML in backend with HTML frontends

2.8 XML Transformations using XSL and XSLT

XSLT converts XML into:

- HTML pages
- XHTML
- PDF (via formatting objects)
- Plain text
- Other XML structures

Example XSLT transforming XML to HTML

```
<xsl:template match="/">
<html>
<body>
<h1><xsl:value-of select="student/name"/></h1>
</body>
</html>
</xsl:template>
```

2.9 DOM and SAX Parsing

DOM Parser

- Loads entire XML into memory

- Easier to navigate
- Suitable for small/medium XML files

SAX Parser

- Reads XML sequentially
- Event-based parser
- More memory-efficient

2.10 Applications of HTML, XML, XHTML in Modern Web Systems

HTML Applications

- Websites
- Web applications
- Mobile apps (WebView)
- CMS systems

XML Applications

- Configuration files
- Financial data systems
- IoT message formats
- Web services
- E-commerce business interchange

XHTML Applications

- Early mobile browsers
- Strict corporate portals
- Embedded browser systems

2.11 Advantages and Disadvantages

HTML Advantages

- Browser-friendly
- Easy to learn
- Supported everywhere

HTML Disadvantages

- Poor data structuring

XML Advantages

- Highly structured
- Machine-readable
- Extensible
- Widely used in enterprise systems

XML Disadvantages

- Verbose
- Not suitable for visual content

XHTML Advantages

- Clean, strict markup
- Consistent browser rendering

XHTML Disadvantages

- Too strict for casual developers
- Lower adoption after HTML5

2.12 Real-World Case Studies

Case Study 1: XML in Banking Systems

Banks exchange data using XML schemas for loan processing, KYC, account metadata.

Case Study 2: XHTML in Early Mobile Browsers

WML/XHTML-MP powered older Nokia browsers.

Case Study 3: HTML5 and XML Integration i

2.3 Understanding the Nature of HTML

2.3.1 Purpose of HTML

- Used for creating web pages.
- Focuses on presentation, structure, and linking of pages.
- Predefined tags such as <p>, <h1>, <table>, etc.

2.3.2 Characteristics

- Not case-sensitive.
- Flexible syntax (missing end tags may still work).
- Primarily designed for browsers.

2.3.3 Document Structure

1. **Document Type Declaration**

   <!DOCTYPE html>
2. **HTML element**

   <html>...</html>
3. **Head section**

   Contains metadata, title, links, styles.
4. **Body section**

   Contains visible content.

2.4 Understanding XML

2.4.1 XML Purpose

- Designed for storing, transporting, and structuring data.
- Does not define presentation.

2.4.2 Characteristics

- Case-sensitive.
- User-defined tags.
- Strict syntax rules.
- Focuses on data interoperability.

2.4.3 XML Document Structure

- **Prolog** (optional)

  Example: <?xml version="1.0" encoding="UTF-8"?>
- **Root element**

  Only one root node allowed.
- **Child elements**

  Structured hierarchically.
- **Attributes**

  Provide additional information.

Example: Basic HTML5 Structure

```
<!DOCTYPE html>
<html>
<head>
   <title>Sample HTML Page</title>
</head>
<body>
   <h1>Welcome</h1>
   <p>This is a sample HTML document.</p>
</body>
</html>
```

2.4.4 Well-Formed XML Rules

- Proper nesting
- Case sensitivity
- Mandatory closing tags
- One root element
- Quoted attribute values

2.5 Understanding XHTML

2.5.1 Why XHTML?

HTML allowed sloppy coding, which created rendering issues. XHTML enforces discipline by combining HTML with XML's strictness.

2.5.2 Features of XHTML

- Uses HTML elements but with XML rules.
- Requires:
    - Lowercase tag names
    - Proper nesting
    - Mandatory closing tags
    - Quoted attribute values
    - <!DOCTYPE> declaration

2.5.3 Advantages

- Consistency across browsers and devices.
- Easier to parse by XML processors.
- Better code quality.

2.6 Comparison of HTML, XML, and XHTML

Feature	**HTML**	**XML**	**XHTML**
Purpose	Web page creation	Data storage/transport	Hybrid (HTML + XML)
Tags	Predefined	Custom	Predefined (HTML) but strict
Syntax	Flexible	Strict	Very strict
Case-sensitive	No	Yes	Yes
Closing tags	Sometimes optional	Required	Required
Structure	Loose	Structured	Well structured
Used for	Display	Data	Display + interoperability

2.7 Relationship Between the Three

The three languages complement each other:

- HTML focuses on how information appears.
- XML focuses on what the data means.
- XHTML aims to close the gap by offering structured HTML that is compatible with XML parsers.

Thus, XHTML can be considered a transitional bridge between HTML and XML.

2.8 Applications

2.8.1 HTML Applications

- Websites
- User interfaces
- Web forms

2.8.2 XML Applications

- Configuration files
- Data exchange (SOAP, RSS, etc.)
- Mobile applications
- Office document formats

2.8.3 XHTML Applications

- Standard-compliant websites
- Mobile web (early WAP/XHTML-MP)
- Systems requiring strict parsing

2.9 Case Study: HTML vs XML in Web Services

- **HTML Example:** Displaying product details on a website.
- **XML Example:** Sending product data between two systems using APIs.

This demonstrates the presentation vs. data distinction more clearly.

2.10 Summary

The chapter presented the basis of three of the most important markup languages namely presentation in HTML, structured data in XML, and an intermediate one in XHTML between strictness in HTML and XML. These basic concepts are the foundation of more advanced standards such as the validation of documents, schemas and transformation and sophisticated web technologies.

.

Chapter 3: Web Page Styling

3.1 Introduction to Web Page Styling

Web page styling- it is simply the process of enhancing the graphic appearance and design of a web page and blunt it out to be more appealing and easier to use.. The primary tool for achieving this is Cascading Style Sheets (CSS).

A webpage or a web page is a document typically written in the language of Hyper Text Markup (HTML) which is accessible with the usage of the Internet or other network with the help of an Internet browser. A web page is opened with typing a URL address and could have any type of text, graphics, and links to other web pages and files. The current page is the sample of a web page. Tim Berners-Lee developed the first web page in 1991 (August 6) in CERN.

Website

A site, a web site or a web site is a central part of a number of web pages, the entire number of which are linked and can be accessed by simply visiting the home page with a browser.

Whose the difference between a web page and a website?

Web site is a central point which is described as having 2 or more web pages. Facebook.com can be used as an example as it is a website, that is, it consists of thousands of various pages.,

3.2 Basics of Web Page Design

3.2.1 Definition of Web design

Web design involves creating the layout of a single web page or an entire website. It can range from producing a visual layout as an image to fully developing the page using graphics, software tools, and programming.

3.2.2 Key Guidelines for Effective Web Design

1. PURPOSE

Good web design will never withhold to the requirements of the user. Do your web visitors seek information, entertainment, some form of communication, or do they want to make an acquisition to your business? Every single page of your site must have an established purpose and complete a particular need among the users of your website in the best manner ever.

2.COMMUNICATION

Web users usually want information quickly, so it’s crucial to present content clearly and make it easy to read and understand. Effective web design strategies include organizing content with headings and subheadings and using bullet points rather than long, complex sentences.

3.COLORS

Colors are very much appropriate in the interaction of the users with your site. The complement colors give it a balance in appearance and contrasting colors in reading gives it a better look and is easy to read. Being sparse with bold and vibrant colors in order to emphasize on important parts of a website such as buttons or an area of action. Along with it, the use of white space also contributes to the creation of the smooth and orderly appearance..

4. **IMAGES**

A single picture can express what words sometimes cannot. Using appropriate images on your website supports brand identity and connects with your audience. When high-quality professional photos are not available, consider stock photography. Additionally, infographics, videos, and illustrations often communicate more clearly than written content.

3.3.3 CSS SYNTAX

CSS rules consist of a selector (which HTML element to style) and a declaration block containing one or more declarations. Each declaration includes a property (e.g., color, font-size) and a value (e.g., blue, 16px).

```
h1 {
  color: blue;
  font-size: 2em;
}
```

- **Applying CSS:**
- **Inline Styles:** Applied straightly to an HTML element using the style attribute. Not recommended for bigger projects.
- **Internal Styles:** Defined within a <style> tag in the <head> section of an HTML document.
- **External Stylesheets:** The most common and recommended method, where CSS rules are written in a separate .css file and linked to the HTML document using a <link> tag in the <head>.
- **Core Styling Properties:**
- **Typography:** font-family, font-size, color, text-align, line-height.
- **Layout:** width, height, margin, padding, display, position, float, flexbox, grid.
- **Backgrounds:** background-color, background-image, background-repeat.
- **Borders:** border-width, border-style, border-color.
- **Selectors:**

CSS provides various ways to target specific HTML elements for styling:

- **Element Selectors:** Target all instances of an HTML element (e.g., p, div).
- **Class Selectors:** Target elements with a specific class attribute (e.g., .my-class).
- **ID Selectors:** Target a single element with a unique id attribute (e.g., #my-id).
- **Descendant Selectors:** Target elements that are descendants of other elements (e.g., div p).

Example of CSS

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>
CSS Good morning.html
</title>
<link rel="stylesheet" type="text/css" href="style1.css"
title="Style 3" />
<link rel="alternate stylesheet" type="text/css" href="style2.css"
title="Style 4" />
</head>
<body>
<p>Good morning! </p>
</body>
</html>
```

3.4 CSS Design Features

CSS (Cascading Style Sheets) provides a wide range of tools that allow developers to style and format web pages, turning simple HTML layouts into attractive and engaging interfaces.

Some of CSS Design Features

Styling and Typography:

Colors and Background:

Managing text colors, page backgrounds, background images, and related settings such as size, positioning, and repetition.

Fonts:

Setting the font family, its size, styling options like bold or italic, and the thickness of the text.

Text Properties:

Adjusting how text is aligned, decorated (underline, line-through, overline), transformed (uppercase or lowercase), spaced, and shadowed.

CSS Layout & Positioning Systems

Box Model:

Handling an element's spacing and size through its margins, padding, borders, height, and width.

Display Properties:

Determining how various elements should render, i.e. should they be a block-level or inline element, inline-block or use flex or grid layout modes.

Positioning:

Positioning elements accurately by using the relative and absolute properties, fixed, sticky and using the z-index to control their depth.

Floats and Clear:

Laying things side by side or allowing to wrap text through introduction of float, and use of clear resolves the flow problems.

Flexbox and Grid:

CSS layout tools that are powerful flexbox is used to arrange content across one dimension whereby grid is a structured system that uses two dimensions to arrange the content.

3.5 Styling HTML Documents

CSS are created to style the HTML documents enabling developers to control the visual appearance of the document, i.e., layout, typography, colors, and spacing.

CSS may be integrated into an HTML document in three ways:

Inline CSS: inlinecss Inline CSS is the simplest to create and implores the best when only one or two small adjustments in the canvas of a particular element are required (CSS also includes able to specify target elements).

Code:

```
It is a blue, 16 arrive paragraph.
```

Internal CSS: Internal CSS writes style rules as part of <|human|>Internal CSS is appropriate in styling a single page with maintaining the organizing nature of styles..

Code:

```
<!DOCTYPE html>
```

```
<html>

<head>

    <title>Internal CSS Example</title>

    <style>

        h1 {

            color: green;

            text-align: center;

        }

        p {

            font-family: Arial, sans-serif;

        }

    </style>

</head>

<body>

    <h1>Welcome!</h1>

    <p>This paragraph uses an Arial font.</p>

</body>

</html>
```

External CSS: External CSS, the most widely used method, involves placing all style rules in a separate .css file and linking it to the HTML document via the <link> tag in the <head>, promoting reusability and easier maintenance.

```
<!DOCTYPE html>

<html>

<head>

    <title>External CSS Example</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <h1>Welcome!</h1>

    <p>This paragraph is styled by an external stylesheet.</p>

</body>

</html>
```

styles.css:

```
h1 {
    color: purple;
    text-decoration: underline;
}
p {
    font-size: 18px;
    line-height: 1.5;
}
```

Use Case: This allows centralized styling which enhances the code reuse, as well as maintainability, in the contexts of multiple web pages and any changes towards the CSS will be immediately reflected throughout the site.

- Text Formatting

  Formatting of text arranges and incorporates written work into a more consumable and simpler to read form with font styles, font size, font color, font alignment, font spacing and incorporates structured forms such as paragraphs and lists.

- Popular Text Formatting Charlies.

  Font and hidden size: By adjusting the typeface and the size a person may change the visual appearance of the text.

- **Bold, Italics, and Underline:** It is used to make the emphasis or highlight some words or phrases we consider important.
- **Color**: Color exaggeration of the text in order to increase its visibility or to create meaning.
- **Alignment**: Text may be aligned to the left, right, center or justified as a enhancement of the appearance and text readability.
- **Lists**: Writing down in either set of bullets or a set of numbers to have items that are easier to comprehend.
- **Headings and Subheadings**: To achieve the clear content hierarchy, different sizes and variants of the font are used.
- **Highlighting**: The act of adding a background color into text in order to highlight attention, commonly thought of as an independent feature.
- **How To Find Text Formatting Visuals.**
- Word Processing Software: This can largely be found in the home tab, in other areas such as font and paragraph.
- Web Interfaces: This is displayed in formatting toolbars or context menus, which are shown when the text is selected.

- Web Development (HTML and CSS): This can be controlled via the code, which defines the text styles, colours, alignment and other visual attributes on the web pages.
- **3.6 Knowing the Cass Typic Box Model in CSS.**
- CSS Box Model is one of the concepts of web designing that involves presentation of the HTML elements as boxes in terms of the size, spacing and connection with each other. Each of the elements, be it a text or image and any other element of the HTML is a box that consists of four individual layers:
- **Content**: The deepest part where is the actual information about the element such as text or images. Its dimensions depend on width and height characteristics.
- **Padding**: The open space which is a feature of the content that separates the content and the border of the element. The padding can be padded out on each of the sides separately or padded together on the padding property.
- **Border**: This is a visible layer, which encloses around the content and padding, and may have various styles, thicknesses, and colors.. Borders can be customized on each side using the border property.
- **Margin:** The outermost transparent area that separates the element from other nearby elements. Margins can also be set individually or uniformly with the margin property.

3.7 CSS Cascading and Inheritance

Cascading:

Cascading describes how the browser decides which CSS rules to apply when multiple rules target the same element. The process follows a hierarchy of precedence:

- Origin and Importance: This is the top level of priority, based on where the style comes from—user agent stylesheet, user stylesheet, or author stylesheet—and whether it uses !important. Rules marked !important take precedence over normal rules, and author-defined styles generally override user or browser defaults.
- Specificity: The more specific a selector is, the higher its priority. For example, ID selectors override class selectors, which in turn override element selectors.
- Order of Appearance: If rules have the same origin, importance, and specificity, the one that appears later in the stylesheet or in linked stylesheets takes precedence

Inheritance:

Inheritance is the process by which certain CSS properties applied to a parent element automatically affect its child elements.

- Inherited Properties: Properties like color, font-family, font-size, text-align, and line-height are typically inherited.
- Non-Inherited Properties: Properties such as margin, padding, border, width, and height do not inherit automatically and must be explicitly set.

- Overriding Inheritance: Child elements can override inherited properties by specifying their own values. CSS keywords like inherit, initial, unset, and revert can also control inheritance behavior.

Rule cascading

Rule cascading is the method browsers use to decide which CSS styles take effect when multiple rules apply to the same element, following a hierarchy based on origin and importance, selector specificity, and declaration order.

Example

Imagine a paragraph element (<p>) that has both an external stylesheet and an inline style.

- **External stylesheet:** $p { color: blue; }
- **Inline style:** <p style="color: red;">
- **The result:** The paragraph will be red, not blue, because inline styles have higher importance than external stylesheets.

Style Inheritance:

Style inheritance is the mechanism by which child elements receive style properties from their parent elements. For instance, setting a font or color on <body> in CSS applies that style to all nested elements unless they are overridden. Not all CSS properties are inherited automatically.

For example, layout properties like margin and padding are typically not inherited, while text-related properties like font-family and color.

Parent-to-Child Relationship: Styles applied to a parent element naturally flow down to its child elements. For example, if a <div> has color: blue, all <p> tags inside it will inherit that blue color.

Inherited vs. Non-Inherited Properties: Some CSS properties, such as font-family, font-size, and color, are inherited automatically, while others, like border, margin, and padding, must be explicitly set on each element.

Overriding Inheritance: Styles applied directly to a child element take precedence over inherited styles. For instance, if a <p> is styled with color: red inside a blue <div>, it will appear red.

Default Behavior: When a child element doesn’t have a style defined, it inherits from its parent. If the parent lacks a defined value, the browser checks up the DOM tree until it finds a set value.

Forcing Inheritance: The inherit keyword can be used to make any CSS property inherit its value from a parent element, even for properties that are normally non-inheritable.

Benefits:

- **Simplifies Code:** Applying styles to a high-level element, such as <body>, eliminates the need to repeat the same rules on every individual element.
- **Easier Maintenance:** Changing a style on a parent element automatically updates all descendant elements that rely on inheritance, making style management more efficient.

3.8 Normal Flow Box Layout

Normal flow is the default way HTML elements are laid out on a webpage, based on their source order, before any layout-altering CSS is applied. It consists of two main types: block-level elements, which stack vertically, and inline-level elements, which flow horizontally within lines of text. This creates a predictable structure that is readable even without styling, ensuring content is accessible to users with screen readers or limited browser features.

Block-level elements

- Behavior: Display one after another, starting at the top of the containing block and moving down.
- Examples: <p>, <div>, <h1>, <ul>.
- Formatting: They form a new block, start on a new line, and take up the full width available unless a specific width is set. They can have their width, height, and top/bottom margins and padding defined.
- Formatting context: They participate in a block formatting context.

Inline-level elements

- Behavior: Display horizontally, one after another, starting from the left (in English).
- Examples: <span>, <a>, <strong>.
- Formatting: They flow within the line of text, and a new line is created only when the line of text is full. They respect horizontal margins, borders, and padding, but their top and bottom margins do not have a visual effect in the flow.
- Formatting context: They participate in an inline formatting context.

3.8.1 Basic Box Layout

CSS Box Model is a web design principle, where HTML elements are rendered using browsers as rectangular boxes. The box model is essential in the control of the layout, spacing and look of the items on a webpage.

All the elements are considered as boxes comprising four layers:

- Content Box: This is the deepest layer which contains the content of the element, which may be text, images or other elements. The width and the height properties are used to define the size of the content box. Neither the padding, border nor margin is altered when these properties are adjusted.
- Padding Box: The padding box encircles the content box and it is an empty space that divides the content and the element border. Padding also makes sure that there is not any direct contact of text or image with the border. The padding space also offers the continuity of the

background color of the element. The padding may be individual on each side or even on both sides in the same way.

- Border Box: Borders the content and padding of the element, presenting a graphics display of a dividing delineator;. Borders can be styled in various ways, such as solid, dashed, or dotted, and their thickness and color can be customized. The border's dimensions are added to the total size of the element unless using box-sizing: border-box.
- Margin Box: The outermost layer, the margin, creates space between the element and surrounding elements. Margins are transparent and do not affect the element's background. They can collapse in certain cases, such as when vertical margins of adjacent elements meet.

3.8.2 The display Property

The display property in CSS determines how an HTML element is rendered in the document and how it interacts with other elements. It controls the element's layout behavior, affecting whether it generates a block, inline, or other type of box.

Common Values of display:

- block: The element starts on a new line and takes up the full width available. Examples: <div>, <p>.
- inline: The element does not start on a new line and only takes up as much width as its content. Examples: <span>, <a>.
- inline-block: Behaves like inline elements but allows setting width and height like a block element.
- none: Hides the element completely; it does not occupy any space in the layout.
- flex: Turns the element into a flex container, enabling flexible layouts for its children.
- grid: Turns the element into a grid container, allowing placement of children in rows and columns.
- list-item: Makes the element behave like a list item, typically with a bullet or number.

3.8.3 Margin Collapse

Margin collapse is a CSS behavior where the vertical margins of adjacent block-level elements combine into a single margin instead of adding together. This helps maintain consistent and predictable spacing between elements.

How It Works:

1. Adjacent Sibling Elements:
   - o When two block-level elements are stacked vertically, the larger of the two vertical margins is applied.
   - o Example: If the first element has margin-bottom: 20px and the next element has margin-top: 30px, the total space between them will be 30px, not 50px.

2. Parent and Child Elements:
    - A parent’s top margin can collapse with the top margin of its first child.
    - A parent’s bottom margin can collapse with the bottom margin of its last child.
3. No Collapse Situations:
    - Margins do not collapse if there is padding, a border, or inline content between the margins.
    - Horizontal margins never collapse; this applies only to vertical margins.

Importance:

- Prevents excessive spacing and helps create consistent layouts.
- Essential for understanding spacing behavior in CSS box model layouts.

Key Aspects of the CSS Box Model

- **Width and Height Calculation:** By default, the width and height properties in CSS only apply to the content box. The total size of an element is calculated by adding the content dimensions, padding, and border.
- **box-sizing Property:** The box-sizing property controls how the total width and height of an element are calculated.
    - **content-box (default):** The width and height values apply only to the content area.
    - **border-box:** The width and height include padding and border, making it easier to manage overall element size.
- Margin Collapsing: In certain situations, vertical margins between adjacent block-level elements can collapse. Instead of adding together, the larger margin value is used.
- **Block vs. Inline Elements:** All elements follow the box model, but their layout behavior differs:
    - **Block-level elements** (e.g., <div>, <p>) occupy the full width of their container and stack vertically.
    - **Inline elements** (e.g., <span>, <a>) only take up as much width as their content and flow horizontally within a line. Inline elements have limited control over box model properties like width, height, and vertical margins.

3.9 CSS Box Model Shorthand

CSS Box Model comprises of four layers which are content, padding, border, and margin. Defining related values in one single declaration by way of shorthand properties is a cleaner and better-manageable way of defining many related values.

1. Padding Shorthand:

padding property is to be capable to pad an element on every side simultaneously.

Single Value: In case of giving one single value, it is applied equally to all four sides, including the top, right, bottom and left sides.

Example padding: 10px;

Two values: The first one is used in the top and bottom levels, the second in the left-hand and the right-hand.

Example : padding: 10px 20px;

Three values: The first one is used on top, the second is used on both sides: left and right and the third value is used on the bottom.

Example padding: 10px 20px 15px;

3.10 Summary

Web page styling is the enhancement of webpage visual styles and design of webpages by CSS (cascading style sheets). One of the HTML pages is called a webpage, and a collection of connected webpages is referred to as a site. CSS enables the designer to manipulate colors, layout, fonts, spacing among others..

Chapter 4: XHTML (Extensible HTML)

4.1 INTRODUCTION TO XHTML

XHTML is a markup language that has been developed to structure web pages whereas it is a more stringent and XML-compliant version of HTML. It exists to use the strict rules of syntax of XML on web programming, so that documents are still well-formed and simple to read by XML coders.

Key Differences from HTML

Well-Formed Documents: Every element must be correctly nested and explicitly closed. For instance, <b><i>text</i></b> is valid, while <b><i>text</b></i> breaks proper structure.

Lowercase Tags and Attributes: All tag and attribute names must appear in lowercase, since XML treats uppercase and lowercase characters as distinct.

Quoted Attribute Values: Attribute values must always be wrapped in quotation marks—for example, <a href="page.html"> rather than <a href=page.html>.

No Attribute Minimization: Boolean attributes cannot be shortened; they must be written in their full form, such as checked="checked" or disabled="disabled".

Self-Closing Tags: Elements that contain no content must end with a trailing slash to indicate they are self-closing, like <br /> or <img src="image.jpg" alt="Description" /> instead of <br> or a bare <img>.

Required Elements: A valid XHTML document must include <html>, <head>, <title>, and <body> elements, along with a <!DOCTYPE> declaration and an xmlns attribute inside the opening <html> tag.

Standard XHTML Document Structure :

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>

   <title>Document Title</title>

</head>

<body>

   <p>This is the content of the document.</p>

</body>

</html>
```

4.2 **UNDERSTANDING XML VERSIONS AND THE XML DECLARATION**

XML documents can start with an XML Declaration that gives XML processors important info about the document. If it's there, it has to be the first line.

XML VERSIONS

- The version attribute within the XML declaration designates the particular XML specification the document is intended to conform to.
- **XML 1.0** remains the primary version in use, with extensive support throughout most XML processors and applications.

XML 1.1 though available, is adopted less frequently. It introduces broader Unicode compatibility—accommodating scripts such as Cambodian, Burmese, and Mongolian—and relaxes restrictions to allow certain legacy control characters.

XML DECLARATION

The XML declaration starts with <?xml and ends with ?>, and it's got a few attributes, including version, which is required and usually set to "1.0".

Encoding:

Encoding is an optional attribute that specifies the document's character encoding, like "UTF-8" or "ISO-8859-1". If it's not there, UTF-8 or UTF-16 is usually assumed.

Standalone:

Standalone's an optional attribute that says if the document's self-contained ("yes") or needs external stuff like a DTD ("no"). Defaults to "no" if not specified.

Example

<?xml version="1.0"?>

<text>Good morning </text>

Key Points:

The XML declaration is not an element and therefore it does not have a closing tag.

The qualities in the declaration are supposed to be presented in the designated sequence; first form, then encoding and lastly standalone where available.

The names of the attributes, which are always in lowercase are case-sensitive together with their values.

You can either single or use a doubled quotation mark to enclose the attribute values.

XML namespaces have one dimension less than the five dimensions of the WFN.

4.3 AN OVERVIEW OF XML NAMESPACES

XML namespaces are useful in preventing similarities in names in the XML documents, in particular where one wants to incorporate elements as well as attributes, which are of varying sources. They ensure that names are not repeatable, although they have the same local name.

<description>I've been spending the weekend at the

<a href=http://airrace.org>Reno Air Races</a>.

Key Concepts:

- Namespaces use URIs to uniquely ID elements and attributes, preventing clashes.
- Declared using xmlns attribute, linking a URI to a prefix or default namespace.
- Prefix declaration: xmlns:prefix="URI" (e.g., prefix:element).
- Default namespace: xmlns="URI" applies to elements without a prefix.
- Scope applies to the element and its descendants, unless overridden.

Examples

```
<bookstore xmlns:bk="http://www.example.com/books">
  <bk:book>
   <bk:title> Harry Potter </bk:title>
    <author> william shakespeare </author>
  </bk:book>
  <magazine xmlns="http://www.example.com/magazines">
    <title>National Geographic</title>
    <issue>December 2025</issue>
  </magazine>
</bookstore>
```

4.4 JavaScript, XML, and the Ajax Model

AJAX (Asynchronous JavaScript and XML) constitute a collection of methods, which helps the web applications to trade with a server, asynchronously. This allows only relevant updates of the content of web pages to be made without complete re-loading of the pages and enhances responsiveness and interactivity.

Although XML was the main choice of the data format initially, modern AJAX based applications tend to use JSON because it can be convenient and integrated with JavaScript.

JavaScript self-inclusive of XML or other formats AJAX normally consists of:

- XMLHttpRequest Object or Fetch API:JavaScript makes asynchronous XHR or Fetch API HTTP requests and allows background communication with the server.
- Request Transmission: The script identifies the method to use in the HTTP request, the endpoint to access, and any data included with it (e.g. in a form or as JSON) and makes the request.

- Response Handling: When a response is received, the JavaScript is used to parse the XML through the DOM or converts responses in a form of a JSON object, which can be manipulated as a JavaScript object.
- DOM Manipulation: JavaScript is used to manipulate the desired elements of the DOM based on the data obtained to refresh content without having to reload the page..

Example

```
// Function to load XML data asynchronously
function loadXMLData() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      // Process the XML response
      var xmlDoc = this.responseXML;
      var items = xmlDoc.getElementsByTagName("item");
      var output = "<ul>";
      for (var i = 0; i < items.length; i++) {
        output += "<li>" + items[i].getElementsByTagName("title")[0].childNodes[0].nodeValue +
"</li>";
      }
      output += "</ul>";
      document.getElementById("content").innerHTML = output;
    }
  };
  xhttp.open("GET", "data.xml", true); // Requesting an XML file
  xhttp.send();
}
// Call the function to load data when needed
// loadXMLData();
```

4.5 AN OVERVIEW OF DOM-BASED XML PROCESSING

XML processing written in DOM describes an XML document as a tree resource within memory and it can be accessed and manipulated programmatically as its content, structure, and style.

XML processing as an XML document is in the form of a tree hierarchy of nodes which is XML-based. The elements, attributes and text are represented as single nodes, and parentchild relationships are single child and single parent.

The loading of an XML document with a DOM parser creates this whole tree in memory giving full and immediate access to all the portions of the document.

The DOM specification identifies a standard API which supporting:

- **Traversal:** Moving through nodes in the tree.
- **Data Retrieval:** Reading the contents of elements and attributes.
- **Alteration:** Modifying node values or structures.
- **Insertion:** Adding new nodes to the tree.
- **Removal:** Deleting nodes from the hierarchy.

As a W3C standard, the DOM ensures platform- and language-independent manipulation techniques.

Advantages:

- **Direct Access:** Any node can be reached without sequential processing.
- **Comprehensive Manipulation:** Enables complex updates to both structure and data.
- **Versatility:** Suitable for applications with intensive document interaction requirements.

Disadvantages:

- **Memory Overhead**: large document has to be in memory and this is costly with large XML documents.
- **Slower intialization** : Parsing to get into the full tree is slower than streaming parsers such as SAX where partial processing is needed.

The fourth paper will be Fromatus et sax, an event-based XML parser.

SAX or the Simple API of XML gives an event based mechanism of processing XML documents. Contrary to a tree-based parser (like DOM Document Object Model): the XML file is loaded into memory as a hierarchical structure where the complete structure is in memory; in SAX, the data is read in sequence and an event sent whenever a particular structure of an XML is reached..

Key characteristics of SAX:

- Event-Based Processing: Instead of constructing a tree, SAX notifies the application as parsing events occur, such as when an element begins or ends, when text content is encountered, or when a processing instruction appears.
- Linear Parsing Flow: The document is scanned from start to finish, with events emitted in the order elements appear.
- No Full Document Representation: Because SAX never builds a complete abstract syntax tree (AST), it is extremely efficient for handling very large XML files or working in memory-constrained environments.

- Callback Architecture: Applications provide handler classes (e.g., a Content Handler) that define how to react to the incoming events. Whenever the parser encounters a particular XML structure, it calls the corresponding handler method.

Advantages:

- Low Memory Footprint: Suitable for massive XML files since the parser doesn't need to load the entire document.
- High Performance: Often faster than DOM when only specific segments of data must be processed.
- Efficient Resource Use: Works well on systems with limited processing or memory capacities.
- Disadvantages:
- Here is a refined version of your text with alternate sentence phrasing while preserving the meaning. I've varied structure, vocabulary, and flow without changing the technical content.

Disadvantages:

- Read-Only by Design: SAX focuses on reading streams of XML and isn't intended for altering document structure.
- Manual State Tracking: Because events are delivered in a linear sequence with no built-in structural context, developers must maintain their own state to keep track of where they are in the document.

Common SAX events include:

- Start Document(): Invoked when parsing begins.
- End Document(): Invoked when parsing completes.
- Start Element(uri, local Name, qName, atts): Fired when an opening tag is encountered.
- End Element(uri, local Name, qName): Fired when the corresponding closing tag appears.
- characters(ch, start, length): Called when text content inside an element is read.
- Processing Instruction(target, data): Triggered upon encountering an XML processing instruction**.**

4.7 XML Transformation Methods

XML transformations are most commonly performed using Extensible Stylesheet Language Transformations (XSLT). XSLT is a specialized language designed to convert XML documents into different formats, such as HTML, plain text, or even another XML document.
Key concepts and methods include:

1. XSLT Stylesheets: is based on stylesheets XML documents, themselves, which describe the form of transformation of the source XML. Under these stylesheets, templates are designed to match the elements or attributes in the input XML, normally by using XPath expressions.
Once this matching has occurred the template will specify how this content will be aligned and formatted in the output that will be created..

Example XSLT Stylesheet Structure:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <!-- Rules for transforming the entire document -->
  </xsl:template>.
  <xsl:template match="element_name">
    <!-- Rules for transforming a specific element -->
  </xsl:template>
</xsl:stylesheet>
```

2.**XSLT Processor:**

An XSLT processor is a processor which takes as input an XML document and an XSLT stylesheet.

It deciphers the transformation rules and the stylesheet and transforms the source XML to the output format as required.

There are a number of XSLT processors:

- **Inbuilt Processors**: Some web browsers have inbuilt support of XSLT, and it can be used to render XML documents by using the corresponding stylesheets.
- Libraryz Libraryz zuymerzibrary Java (through javax.xml.transform) .NET (through System.Xml.Xsl), and Python have libraries that support users to execute XSLT transformations programmatically.
- ID-Editors Software XML editors and Integrated Development Environments usually come with integrated XSLT engines which are used to enable users to test, preview and run transformations directly with the software.

XPath:

XPath (XML Path Language) is very essential in XSLT.

In the context of XSLT stylesheets, it is the tool that performs navigation between the XML structure and the specific selection of the node; i.e. an element or attribute or text.

Based on XPath expressions, it is possible to specify the specific elements of the data that the transformation is to act upon.

4.Transformation Process:

XML Document, which is to be undergone (Input) XML Document The original XML material that is to be transformed.

XSLT Stylesheet: It is the set of rules of the transformation depending upon which the source XML should be processed

- XSLT Processor: It is the part which implements the rules of the stylesheet to the XML data.
- Output: This is the final output which is the generated result, sometimes in a different format like HTML, a new XML file or plain text.

Beyond XSLT

Even though XSLT is the most popular technology in the case of general-purpose XML transformations, other options are deployed in some of the more specific cases:

LINQ to XML (.NET): This is a code-based framework that allows the creation, querying, and manipulation of XML through C# or Visual Basic, which is best suited to its use in transformation within an application.

XQuery Update Expressions: This is normally used in the XML-efficient databases like Db2 to conveniently update and manage the XML data that is stored in XML columns.

Simple Transformations (SAP): A transformation language used in the SAP ecosystem wherein the developer can transform ABAP data structures as XML and vice versa

4.8 Selecting XML Data: XPath

XPath is an expression-based language used to traverse XML document trees and retrieve nodes or node sets based on names, attributes, hierarchy, position, and other structural criteria.

Key Concepts and Syntax:

Path Expressions: XPath locates nodes by using path-style expressions that specify where they appear in the XML structure.

/:Indicates that the selection should start at the root node.

//:Searches the entire subtree from the current node for nodes that fit the criteria.

. Refers to the node you're presently positioned at.

nodename: Retrieves every element that matches the given tag name.

@:Used to access attributes of an element.

- [predicate]: Filters nodes based on a condition within the square brackets. Predicates can include functions, comparisons, and logical operators.

- **Examples of XPath Expressions:**

- /bookstore: Selects the root element named "bookstore".

- /bookstore/book: Selects all "book" elements that are direct children of the "bookstore" element.
- //title: Selects all "title" elements anywhere in the document.
- //book[price > 35]: Selects all "book" elements where the "price" child element has a value greater than 35.
- //title[@lang='en']: Selects all "title" elements that have a "lang" attribute with the value "en".
- //book[position()=1]: Selects the first "book" element.
- //book[last()]: Selects the last "book" element.

Applications:

- XPath plays a key role in several XML-related technologies, such as:
- Many XML tools and standards rely heavily on XPath, including:
- XPath underpins a variety of XML technologies.

4.9 JAVASCRIPT AND XML: AJAX

Ajax (Asynchronous JavaScript and XML) is a technique that allows web pages to **retrieve data from the server asynchronously**—without reloading the entire page.

Key Ideas

- Uses **JavaScript** to send background HTTP requests.
- Uses **XMLHttpRequest (XHR)** or **fetch()** to communicate with the server.
- Server can return **XML, JSON, HTML, or text** (JSON is most common today).
- Enables dynamic, interactive web applications.

Basic Ajax Workflow

1. JavaScript creates an **HTTP request**.
2. Request sent to the server **asynchronously**.
3. Server processes the request and returns data (XML/JSON).
4. JavaScript updates the DOM dynamically.

Example Using XML Http Request

```
let xhr = new XML Http Request();
xhr.open("GET", "data.xml", true);

xhr.onload = function() {
  if (this.status === 200) {
    let xml = this.responseXML;
    console.log(xml);
  }
};
```

```
xhr.send();
```

Example Using fetch() (modern)

```
fetch("data.xml")
  .then(response => response.text())
  .then(str => (new window.DOMParser()).parseFromString(str, "text/xml"))
  .then(xml => console.log(xml));
```

4.10 DOM-Based XML Processing

DOM-based XML processing uses the **Document Object Model (DOM)** to represent XML documents as a **tree structure** in memory, allowing JavaScript to navigate and manipulate them.

Key Ideas

- The XML file is parsed into a **DOM tree**.
- Each element becomes a **node**.
- JavaScript can **traverse, read, modify, or create nodes**.

Core DOM Methods for XML

Operation	Method
t elements by tag	`tElementsByTagName()`
cess node value	`de.childNodes[0].nodeValue`
eate new element	`eateElement()`
pend nodes	`pendChild()`
rse XML string	`MParser()`
rialize DOM back to text	`LSerializer()`

Example: Reading XML with DOM

Assume the XML:

```
<books>
  <book>
    <title>JavaScript Guide</title>
  </book>
</books>
```

JavaScript DOM processing:

```
let xml = xhr.responseXML;
let titles = xml.getElementsByTagName("title");
console.log(titles[0].textContent); // "JavaScript Guide"
```

Creating and Adding New XML Nodes

```
let newBook = xml.createElement("book");
let newTitle = xml.createElement("title");
newTitle.textContent = "New Book";
newBook.appendChild(newTitle);
xml.documentElement.appendChild(newBook);
```

4.11 Event-Oriented Parsing: SAX

What is SAX?

SAX (Simple API for XML) is an event-driven XML parsing model. Unlike DOM, SAX does **not** load the whole document into memory. Instead, it reads XML **sequentially** and triggers events when it encounters elements, attributes, text, etc.

Why Use SAX?

- Efficient for large XML documents
- Low memory usage
- Suitable for streaming data

How SAX Works

The parser generates events:

- startElement(name, attributes)
- characters(text)
- endElement(name)

Your handler code reacts to these events.

Example Concept

```
<book>
  <title>XML Guide</title>
</book>
```

Events fired:

1. startElement("book")
2. startElement("title")
3. characters("XML Guide")
4. endElement("title")
5. endElement("book")

4.12 Transforming XML Documents

Transforming XML means converting an XML document into:

- Another XML format
- HTML
- Text
- JSON
- Or any structured output

Most transformations use **XSLT** + **XPath**.

4.13 Displaying XML Documents in Browsers

Browsers can display XML by:

1. **Default tree view**
2. **Using CSS** (simple formatting)
3. **Using XSLT** (transform XML to HTML)

Example (link XSL in XML):

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Browsers transform XML → HTML automatically.

4.14 Summary

This chapter introduced the XHTML and XML technologies, their syntax rules, methods of processing XML, and their role in modern web development—especially in conjunction with JavaScript and AJAX. It also covers XML namespaces, XPath, XSLT, and how browsers handle XML.

Chapter 5 Server-Side Programming Java Servlets

A servlet **is a** Java program that runs on a web server **and is used to handle** HTTP requests **and generate** dynamic web responses **(like HTML, JSON, etc.). It is part of the** Jakarta EE (formerly Java EE) **web technology stack.**

A.**Servlet=Server+Applet**
It is a **server-side component** that processes web requests.

5.1 Servlet Architecture Overview

Servlets are the Java programs, which are used on the Java-enabled web server or application.

server. They are provided to process the request and deal with the request obtained against the web server.

send out a response, and subsequently send back a response to the web server.

The characteristics of the Servlets are the following:

- Servlets can be utilized at the server side.
- The servlets can be used to process complicated requests received on the web server.
- Servlets lie on the Advanced Java tree which is utilized in the development of dynamic web.
- applications. These are robust and highly scalable whose main application is in the creation of server-side.
- applications. Had we gone somewhat further back in time we should have been able to witness the fact that prior to the

CGI (Common Gateway Interface) was employed as an introduction of servlets. Among several indigenous

Most common tasks that a servlet can be done include dynamically carrying out client requests and client responses. Other functions that can be done effectively by a servlet are:

- Is capable of managing/controlling the flow of applications easily.
- Appropriate to apply business logic.
- Is able to efficiently distribute the load on the server.
- POC - large scale web generation.
- Server and Client Response.

Also can be used as an interceptor or filter of a certain group of requests.

The image itself can be represented as Servlet Architecture being as depicted below;

Types of Servlet

Generic servlets: These are the servlets which offer functionality in the implementation of a True to type servlet.

1. servlet. It is a generic class out of it all the customizable servlets are derived. It is
2. protocol-independent and supports FTP, HTTP and SMTP protocols. The class
3. javax.servlet.Servlet has been used and it only has 2 methods namely init to initialize and allocate.

To the servlet, memory and to deallocate the servlet, destroy.

HTTP Servlets: This is a protocol dependent servlet, which supports the HTTP.

request and response. It is normally applied in the development of web applications. And has two of the most used

procedures - doGET and doPOST equal as far as they serve their different purposes.

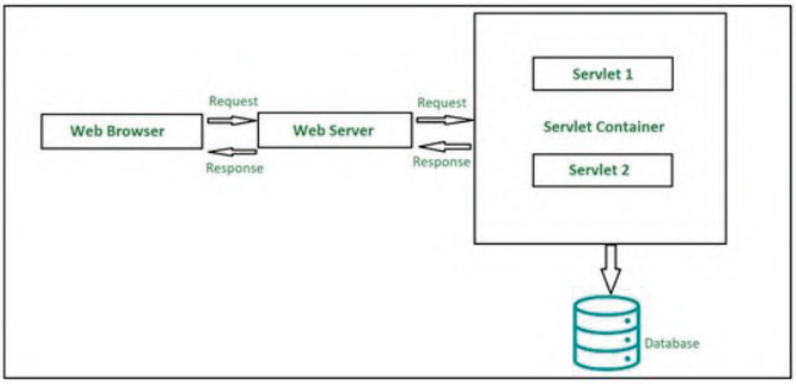
We can use there are three possible ways in which we can use to create a servlet:

- The application of Servlet Interface.
- Extending Generic Servlet
- Extending HTTP Servlet
- Servers of Servlet Architecture.

Servlet Architecture would hold the business logic to carry out all the requests by client.

The high level architecture diagram of servlet will be stated below. And perchance in short, how does each?

part comes into action in creation of a servlet..



1. Client

The customer represented by the architecture above is that of the web browser and it mostly functions as.

a medium which transmits the HTTP requests to the web server and web server creates a.

some processing in the servlet and the client further processes the response, response based.

2. Web Server

Primary task of a web server is to handle the services sent by the users in terms of the request and response.

time and retain the manner in which a web user has the capacity to access the files that is being hosted over the server.

The server, which we are referring to in this case is a software that regulates access to a centralized.

network resource or service. Exactly, there are two kinds of webservers:

- Static web server
- Dynamic web server

3. Web Container

Another common element of servlet architecture that is known as web container does it.

it is communicating with the servlets. Two major activities of web container are:

- Service life cycle management.
- URL mapping

Web container is positioned at the server-side and it is processing and taking care of all the requests which are received.

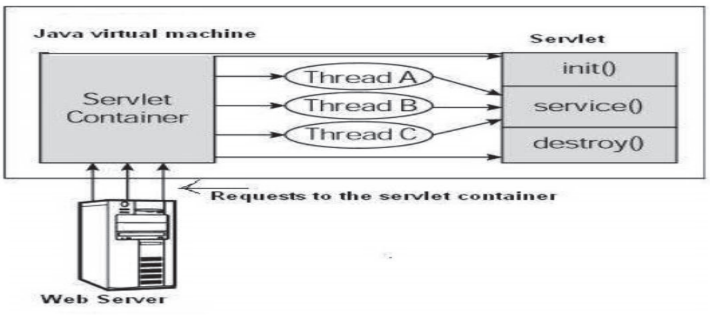
either via the servlets or some JSP pages or even any other file system.

What is the process of Servlet Request?

All the servlets are expected to override the following 3 methods namely:

1. init(): To initialise/instantiate the servlet container.
2. service(): This is an interface that serves as a mediator between the HTTP request and the.
   the appropriate business logic to fulfill such a request.
3. destroy(): This is a method employed to release the memory that was used to give the servlet.

These procedures are applied in processing the user request.



Sample Code

// Import required java libraries

import java.io.*;

```
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloWorld extends HttpServlet {
  private String message;
  public void init() throws ServletException {
    // Do required initialization
    message = "Hello World";
  }
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

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

    // Actual logic goes here.
    PrintWriter out = response.getWriter();
    out.println("<h1>" + message + "</h1>");
  }
  public void destroy() {
    // do nothing.
  }
}
```

The steps of the request being passed through a servlet comes second and this can be noted in the architecture diagram:

- The request is over sent by the client.
- The web server accepts the request and the web container receives the request.
- To have the address of the service, the web controller follows the tracks of web.xml file corresponding.
- However, the above process must have been completed by the time the servlet is instantiated and initialized. init() method is invoked in order to initialize the servlet.
- Public service() method is invoked by the calling of public service() method by passing servlet request and response object.

- Container : Under the second step, the public service method type casts the Servlet Request and the Servlet Response objects into the Http Servlet Request and Http Servlet Response objects.
- The public service is invoked in now protected service () method.
- The secured service() way sprouts the request to the proper handler way according to the basis. on the type of request.
- Upon shutting down of servlet container, all the servlets are unloaded and destroy() method is called to all servlets each initialized servlet.

Advantages

- The best attribute of a servlet is that they do not rely on server architecture, and they are relatively compatible with any of the web servers.
- Servlets are another protocol-independent supporting FTP, HTTP, SMTP, etc. protocols to the one fullest.
- Until they are destroyed manually, servlets could be stored in the memory assisting the processing of a number of them.
- requests over time. Also, in cases where a database connection is done it can be used to facilitate process.
- Multiple requests of a database in the same database session.
- Company The servlets share the similarity of Java, which is portability and are, in turn, compatible with almost any web server.
- The servlets are first turned into the byte codes and then the servlets are then executed, thereby assisting in enhancement of the same processing time.

Disadvantages

- Servlet design may be rather tedious.
- The servlet has to be designed with the exception that it is not thread safe.
- Coding a servlet requires more skills to the developers.

5.2 A “Hello World!” Servlet

```
@WebServlet("/hello")
public class HelloWorldServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h1>Hello World!</h1>");
  }
}
```

- Deployment Steps
- Output Screenshot

- Explanation of URL Mapping, doGet(), and PrintWriter

5.3 Servlets – Generating Dynamic Content

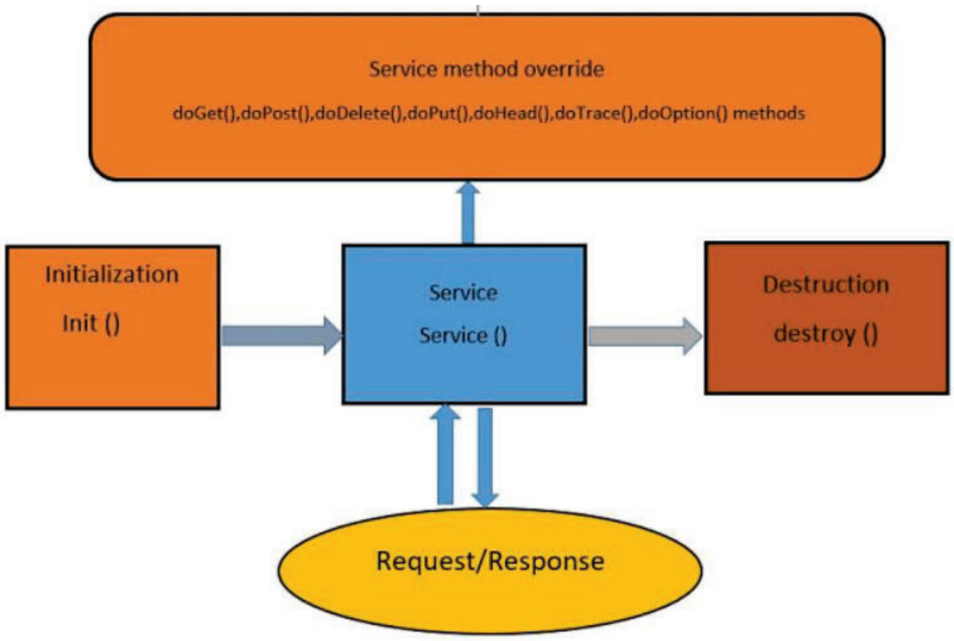
- A **Servlet** is a Java program that runs on the server and responds to client (browser) requests.
- Servlets mainly generate **dynamic web content**, meaning the output changes based on user input, time, or data from a database.
- They run inside a **Servlet container** (e.g., Tomcat), which manages requests, responses, lifecycle, and security.
- When a browser sends a request, the container calls the servlet's **service() → doGet() / doPost()** methods.
- Inside these methods, servlets create dynamic output using **HTML + Java logic**.
- A servlet uses the **HttpServletResponse** object to write the output using a **PrintWriter**.
- Example dynamic data: today's date, student details, login results, database records, etc.
- Servlets support reading user input through **HttpServletRequest** (form fields, query strings).
- They generate output at runtime, so each user can see **personalized content**.
- They are faster, portable, and more secure than CGI because they use Java multithreading.

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

import java.util.Date;

public class HelloServlet extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)

        throws ServletException, IOException {

    res.setContentType("text/html");

    PrintWriter out = res.getWriter();

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

     out.println("<h3>Welcome to Dynamic Content!</h3>");

    out.println("<p>Current Date & Time: " + new Date() + "</p>");

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

  }
```

}

5.4 Servlet Life Cycle

- **Stages:** Load → init() → service() → destroy()
- **Lifecycle Diagram**
- **Code Example** demonstrating init(), doGet(), destroy()



1. Loading the Servlet Class

- When the server starts or when the first request comes, the container **loads the servlet class** into memory.
- This happens only once.

2. Creating an Instance

- The container creates **only one object** of the servlet by calling its constructor.
- This object handles all user requests using threads.

3. Initialization – init()

- The container calls the **init()** method only once.
- Used to load configurations, database connections, or resources required for the servlet.
- It prepares the servlet to handle requests.

4. Request Handling – service()

- For each client request, the container calls the **service()** method.
- It decides whether the request is **GET or POST** and calls doGet() or doPost() accordingly.
- This step executes many times – once per request.
- This is where **dynamic content** is generated.

5. Destruction – destroy()

- When the server shuts down or servlet is removed, the container calls **destroy()** only once.
- Used to close database connections, files, or release resources.

5.5 Parameter Data

5.5.1 Parameter Data and Query Strings

- URL parameters: ?name=John&age=25
- Access in servlet: request.getParameter("name")

5.5.2 Servlets and Parameter Data

- Methods:
  - getParameter(String name)
  - getParameterValues(String name)
  - getParameterMap()

5.5.3 Forms and Parameter Data

HTML Form Example:

```
<form action="greet" method="post">
  Name: <input type="text" name="username">
  <input type="submit">
</form>
```

Servlet Example:

```
String name = request.getParameter("username");
out.println("<h2>Hello, " + name + "!</h2>");
```

5.6 Sessions

5.6.1 Creating a Session

- A **session** is a way to store user information (variables) across multiple web pages.
- Since **HTTP is stateless**, a session helps the server remember the user’s activities during a visit.

- A session begins when a user first requests a web page, and the server assigns a **unique Session ID**.
- This Session ID is usually stored in a **cookie** or passed through **URL rewriting**.
- Servers create sessions using methods like HttpSession session = request.getSession(); in Java Servlets.
- Developers can store data in the session using session.setAttribute("name", value);.
- The stored information stays available until the user logs out or the session times out.
- Sessions are used for tasks like **login management**, **shopping carts**, and **personalized pages**.

// Creating a session

HttpSession session = request.getSession();

// Storing data

session.setAttribute("username", "Sudha");

// **Retrieving data**

String name = (String) session.getAttribute("username");

Uses of Session:

- Login and authentication management.
- Shopping cart in e-commerce websites.
- User preferences and personalization.

- Temporary storage of user data across multiple pages.

5.6.2 Storing and Retrieving Attributes

session.setAttribute("user", "John");

String user = (String) session.getAttribute("user");

5.6.3 Session Termination

session.invalidate();

Use Cases:

- Shopping cart
- User login management

Session Flow Diagram:

Client → Servlet Container → HttpSession → Attributes → Response → Client

5.7 Cookies

Definition

A **cookie** is a small piece of data sent from a **web server** and stored on the **client's browser**. Cookies are used to **identify users**, **store preferences**, and **maintain sessions** between multiple requests, especially when HTTP is stateless.

Characteristics of Cookies

Feature	**Description**
Storage	Client-side (browser)
Size Limit	Usually up to 4 KB per cookie
Lifetime	Can be temporary (session) or persistent (fixed expiry date)
Security	Less secure than server-side sessions
Scope	Can be restricted by domain and path

Types of Cookies

1. **Session Cookies**
   - Stored temporarily in browser memory.
   - Deleted when the browser is closed.
   - Example: Tracking login status during a session.
2. **Persistent Cookies**
   - Stored on disk with an **expiry date**.
   - Used for remembering user preferences, auto-login, or analytics.

Creating a Cookie in a Servlet

```
// Import required packages
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;
// Create a cookie object
Cookie userCookie = new Cookie("username", "JohnDoe");
// Set cookie lifespan (optional, in seconds)
userCookie.setMaxAge(24*60*60); // 1 day
// Add the cookie to response
response.addCookie(userCookie);
```

Explanation:

- new Cookie("name", "value") – Creates a cookie.
- setMaxAge() – Sets expiry (if not set, cookie is session-only).
- response.addCookie() – Sends cookie to the client browser.

Reading Cookies in a Servlet

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for (Cookie c : cookies) {
        if (c.getName().equals("username")) {
            String value = c.getValue();
            out.println("Welcome back, " + value + "!");
        }
    }
}
```

Explanation:

- request.getCookies() retrieves all cookies sent by the client.
- Iterate through cookies to find a specific cookie by name.

Deleting a Cookie

```
Cookie cookie = new Cookie("username", "");
cookie.setMaxAge(0); // Deletes the cookie
response.addCookie(cookie);
```

Explanation:

- Set the cookie value to empty and lifespan to 0 to delete it from the client browser.

Uses of Cookies

- Tracking user sessions without server-side storage.
- Storing user preferences (theme, language).
- Maintaining login status across multiple visits.
- Analytics and personalization.

Cookies vs Sessions

Feature	**Cookies**	**Sessions**
Storage	Client-side	Server-side
Lifespan	Session or persistent	Until session invalidates
Security	Less secure (can be stolen)	More secure (server-controlled)
Storage Capacity	Small (~4 KB)	Can store complex objects
Use Case	User preferences, tracking	Login sessions, sensitive data

Flow of Cookies

Explanation:

- The server sends cookies in the HTTP response.
- Browser stores the cookie and sends it in subsequent requests.

5.8 Data Storage

Data storage in Servlets refers to saving information that can be used across multiple requests or sessions.

- There are **different levels of storage** in web applications:

1. **Request Scope**
   - Data is available only for a single **HTTP request**.
   - Stored using: request.setAttribute("key", value);
   - Example: Passing form data to a JSP page.
2. **Session Scope**
   - Data is available **across multiple requests** from the same user.
   - Stored using: session.setAttribute("key", value);
   - Example: User login details, shopping cart.
3. **Application (Context) Scope**
   - Data is shared **across all users and sessions**.
   - Stored using: getServletContext().setAttribute("key", value);
   - Example: Application-wide counters, configuration settings.
4. **Persistent Storage**

    - Data is stored permanently in **databases or files**.
    - Example: Storing user registrations, orders, or logs.
- **Choosing storage** depends on the **lifetime and visibility** of data.
- Request → Session → Application → Persistent, in increasing scope and lifetime.

5.9 JDBC – Java Database Connectivity

Definition

JDBC (Java Database Connectivity) is an **API in Java** that allows Java applications (including Servlets) to **connect to databases**, execute SQL queries, and retrieve results. It provides a standard interface for interacting with **relational databases** such as MySQL, Oracle, SQL Server, and PostgreSQL.

Key Components of JDBC

1. **Driver Manager** – Manages database drivers and establishes connections.
2. **Connection** – Represents a connection to a specific database.
3. **Statement / PreparedStatement** – Used to execute SQL queries.
4. **ResultSet** – Holds the data returned by a query.
5. **SQLException** – Handles database errors.

JDBC Architecture Diagram:

Steps to Connect to a Database Using JDBC

1. **Load the JDBC Driver**

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

2. **Establish a Connection**

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/mydb", "root", "password");
```

3. **Create a Statement**

```
Statement stmt = con.createStatement();
```

4. **Execute SQL Queries**

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
```

5. **Process Results**

```
while(rs.next()) {
    System.out.println("User: " + rs.getString("username"));
}
```

6. **Close Connections**

```
rs.close();
stmt.close();
con.close();
```

Using PreparedStatement

- **Advantages:** Prevents SQL injection, handles parameters dynamically**Example:**

```
String sql = "INSERT INTO users(username, email) VALUES(?, ?)";
PreparedStatement ps = con.prepareStatement(sql);
ps.setString(1, "JohnDoe");
ps.setString(2, "john@example.com");
ps.executeUpdate();
```

JDBC Drivers Types

Type	**Description**
Type 1	JDBC-ODBC bridge
Type 2	Native-API driver
Type 3	Network Protocol driver
Type 4	Pure Java driver (most common)

Integrating JDBC with Servlets

- Servlets can use JDBC to interact with databases dynamically.
- **Example:** Fetch user details from a MySQL database and display in a web page.

Servlet + JDBC Example:

```
@WebServlet("/showUsers")
public class UserServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
         try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/mydb","root","password");
             Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM users");
             out.println("<h1>User List</h1><ul>");
            while(rs.next()) {
                out.println("<li>" + rs.getString("username") + " - " + rs.getString("email") +
"</li>");
            }
            out.println("</ul>");


            rs.close();
            stmt.close();
            con.close();
        } catch(Exception e) {
            out.println("Error: " + e.getMessage());
        }
    }
}
```

Advantages of JDBC

- Platform-independent API for any relational database.
- Supports dynamic database connectivity in web applications.
- Works seamlessly with Servlets and JSP for dynamic content.

Flow of JDBC in a Servlet

Client Request → Servlet → JDBC Driver → Database → Result Set → Servlet → Response → Client

5.10 SUMMARY

In this chapter introduced the Java Servlets enable dynamic, interactive web applications through request handling, lifecycle management, sessions, cookies, data storage, and database access using JDBC. Servlets form the foundation of modern Java-based server-side development due to their speed, security, scalability, and portability.

References:

Jeffrey C.Jackson, Web Technologies A computer science Perspective Pearson Eucation,2007.

Marjin Haverbeke, Eloquent Java script 4th edition,2024

Uttam K.Roy, Web Technologies Oxford edition,2010.

Dr.Rahul Johari,Web technologies theory & Practical.