BIT2207 : WEB APPLICATION DEVELOPMENT II

PREREQUISITES: WEB APPLICATION DEVELOPMENT I

Lecture Week 1- INTRODUCTION TO WEB APPLICATION DEVELOPMENT II

TERMINOLOGIES

1. Internet The global network of interconnected computers and devices that use the Internet Protocol Suite (TCP/IP) to communicate with each other, enabling the exchange of information and data worldwide.

2. World Wide Web (WWW) A system of interlinked hypertext documents, images, and other multimedia content, accessible via the internet through web browsers. The web is a subset of the internet.

3. Website A collection of web pages organized under a common domain or subdomain, usually accessible via a URL (Uniform Resource Locator).

4. URL (Uniform Resource Locator) A web address that specifies the location of a resource on the internet. It typically includes the protocol (e.g., http, https), domain name, and path to the resource.

5. Domain Name A human-readable label used to identify a website on the internet. It maps to an IP address, making it easier for users to access websites.

6. IP Address (Internet Protocol Address) A numerical label assigned to each device connected to a network, including the internet. It serves as a unique identifier for routing data.

7. DNS (Domain Name System) The system that translates human-readable domain names into IP addresses. It acts as the internet's address book.

8. Web Browser Software that allows users to access and view web pages on the World Wide Web. Examples include Chrome, Firefox, Safari, and Edge.

9. HTML (Hypertext Markup Language) The standard markup language used to create web pages. HTML defines the structure and content of a web document.

10. CSS (Cascading Style Sheets) A stylesheet language used to describe the presentation and formatting of HTML documents, enhancing the visual design of web pages.

11. JavaScript A versatile scripting language used to add interactivity, dynamic behavior, and functionality to web pages. It is primarily used for client-side scripting.

12. Server A computer or software system that provides services or resources to other computers (clients) over a network, such as hosting web pages or handling email.

13. HTTP (Hypertext Transfer Protocol) The protocol used for transferring data on the World Wide Web. It defines how web browsers and servers communicate.

14. HTTPS (Hypertext Transfer Protocol Secure) An extension of HTTP that uses encryption (SSL/TLS) to secure data transmission between a web browser and a server, ensuring privacy and security.

15. Firewall A network security device or software that monitors and controls incoming and outgoing network traffic, acting as a barrier between a trusted network and untrusted networks like the internet.

16. Router A network device that connects different networks and directs data packets between them, facilitating data transmission within and between networks.

17. ISP (Internet Service Provider) A company or organization that provides internet access to individuals and businesses, typically through broadband or other connectivity methods.

18. VPN (Virtual Private Network) A technology that creates a secure and encrypted connection over the internet, allowing users to access the internet as if they were on a private network, enhancing privacy and security.

19. Cookie A small piece of data stored on a user's device by a website, used for tracking and storing information about user interactions and preferences.

20. Firewall A security device or software that monitors and filters incoming and outgoing network traffic to protect a network or computer from unauthorized access and cyber threats.

What is Web Development?

- Web development refers to the process of creating websites, web applications, and other web-based solutions.
- It encompasses both the visual aspects (front-end) and the behind-the-scenes functionality (back-end) of websites.
- Web development is essential for building online platforms, e-commerce sites, social networks, and more.

The Role of Web Developers

- Web developers are the architects and builders of the internet.
- They design, develop, and maintain websites and web applications.

- Developers ensure that websites are functional, user-friendly, and visually appealing

Front-End vs. Back-End Development

- **Front-End Development**
    i. Focuses on the user interface (UI) and user experience (UX).
    ii. Involves HTML, CSS, and JavaScript to create the visual elements users interact with.
- **Back-End Development**
    i. Manages server-side operations.
    ii. Involves databases, server scripting languages, and application logic.

Web 1.0 - The Static Web

- In the early days, the web was static and read-only.
- HTML provided a basic way to display information.
- Websites were simple, text-heavy, and lacked interactivity.

Web 2.0 -The Interactive Web

- Web 2.0 brought interactivity and user-generated content.
- Social media, blogs, and forums emerged.
- AJAX (Asynchronous JavaScript and XML) made web applications dynamic.

Web 3.0 - The Semantic Web

- Web 3.0 aims to provide a more intelligent and interconnected web.
- Focus on data semantics and machine understanding.
- Technologies like RDF and linked data play a role.

Responsive Design

- The rise of mobile devices prompted responsive design.
- Websites adapt to various screen sizes, enhancing user experience.
- CSS frameworks like Bootstrap became popular.

Introduction to Client-Side and Server-Side Programming

Client-Side Programming

- Client-side programming refers to the part of web development where the code is executed on the user's device or in the user's web browser.
- This code is typically written in languages like HTML, CSS, and JavaScript.

- It is responsible for creating the user interface, handling user interactions, and enhancing the user experience.
- Client-side programming focuses on what the user sees and interacts with directly in the browser.

Server-Side Programming

- Server-side programming, on the other hand, involves code that runs on a web server.
- Common server-side programming languages include PHP, Python, Ruby, Java, and Node.js.
- Server-side code is responsible for handling data, managing databases, processing user input, and performing various server-related tasks.
- It often deals with sensitive data and business logic, making it crucial for security and business operations.

Their Respective Roles in Web Applications

Client-Side Programming

- Client-side programming focuses on the presentation and user interface aspects of a web application.
- It controls how web content is displayed, styled, and animated in the user's browser.
- Client-side code can provide instant feedback to users, validate form inputs, and create dynamic user interfaces without requiring a full page reload.
- It is excellent for improving user experience and reducing the load on the server by handling tasks locally.

Server-Side Programming

- Server-side programming manages the backend operations of a web application.
- It is responsible for data storage, retrieval, and manipulation.
- Business logic, authentication, and authorization are often implemented on the server to ensure security and consistency.
- Server-side code generates dynamic content, communicates with databases, and performs tasks that require access to sensitive information.

How They Interact to Create Dynamic Web Experiences

- Web applications often combine both client-side and server-side programming to provide dynamic and interactive experiences.

- When a user interacts with a web application, the client-side code handles the immediate response to that interaction, such as showing or hiding content, providing visual feedback, or making asynchronous requests to the server.
- The client-side code can send requests to the server-side code for data retrieval or processing.
- The server-side code processes these requests, interacts with the database, performs business logic, and sends the necessary data back to the client-side code.
- The client-side code can then update the user interface with the received data, ensuring a seamless and responsive user experience.
- This interaction between client-side and server-side code allows web applications to dynamically generate content, personalize user experiences, and perform complex tasks without requiring constant page reloads.

Client-Side Programming Languages

Overview of Client-Side Programming Languages

- Client-side programming languages are used to create interactive and dynamic user experiences within a web browser.
- These languages run directly on the user's device, enabling the manipulation of web page content without the need for constant communication with the server.
- Common client-side programming languages include HTML, CSS, and JavaScript.

Focus on JavaScript and Its Importance

JavaScript

- JavaScript is one of the most prominent and essential client-side programming languages for web development.
- It is a high-level, versatile, and dynamically typed scripting language.
- JavaScript enables developers to add interactivity, validate user input, and create dynamic content on web pages.
- It is supported by all modern web browsers, making it a universal choice for web development.

Importance of JavaScript

1. **Interactivity** JavaScript allows developers to add interactive elements to web pages, such as buttons, forms, sliders, and menus. This interactivity enhances the user experience.
2. **Asynchronous Operations** JavaScript supports asynchronous programming, enabling the retrieval of data from servers without blocking the main thread. This leads to faster and more responsive web applications.
3. **DOM Manipulation** JavaScript can manipulate the Document Object Model (DOM), allowing developers to dynamically update page content, structure, and styles.
4. **User Interface Enhancements** JavaScript frameworks like React, Angular, and Vue.js simplify the development of complex user interfaces, providing reusable components and efficient rendering.
5. **Form Validation** JavaScript can validate user inputs in real-time, reducing the chance of incorrect data submission and enhancing data quality.
6. **Browser Compatibility** JavaScript is widely supported by browsers, ensuring that web applications work consistently across different platforms.

Benefits and Limitations of Client-Side Scripting

Benefits of Client-Side Scripting

1. **Improved User Experience** Client-side scripting enhances the interactivity and responsiveness of web applications, making them more engaging and user-friendly.
2. **Reduced Server Load** Client-side scripts can handle tasks locally, reducing the need for frequent server requests. This leads to faster page loads and reduced server load.
3. **Real-Time Validation** Client-side scripting can validate user inputs in real-time, providing immediate feedback to users and improving data quality.
4. **Cross-Browser Compatibility** Modern client-side programming languages like JavaScript are supported by most browsers, ensuring compatibility and a consistent user experience.
5. **Rich User Interfaces** Client-side frameworks and libraries enable the creation of rich and complex user interfaces, making it easier to develop feature-rich web applications.

Limitations of Client-Side Scripting

1. **Security Risks** Client-side code is visible to users and can be manipulated, leading to potential security vulnerabilities if not properly secured.
2. **Performance** Excessive use of client-side scripts can slow down web pages, especially on older devices or slower network connections.
3. **Accessibility** Overly complex client-side interactions can create accessibility challenges for users with disabilities.
4. **Search Engine Optimization (SEO)** Search engines may have difficulty indexing content generated by client-side scripts, impacting a site's search engine ranking.
5. **Dependence on Browser** Client-side code's functionality depends on the capabilities and performance of the user's browser.

CLIENT-SERVER MODEL

The client-server model is a fundamental architecture used in computing and networking that describes the relationship between two entities the client and the server. This model is widely employed in various domains, including web applications, file sharing, email, and database management systems. Here's an overview of the client-server model

1. Client

- A client is a device, application, or software that initiates a request for a service or resource from another device or server.
- Clients are typically end-user devices like computers, smartphones, tablets, or IoT devices.
- The client's role is to make requests for specific services or data and process the responses received from the server.

2. Server

- A server is a powerful computer or software application that provides services, resources, or data in response to client requests.
- Servers are designed to be reliable, highly available, and capable of handling multiple client requests simultaneously.
- Common types of servers include web servers, file servers, email servers, and database servers.

Key Characteristics of the Client-Server Model

1. Request-Response Architecture

- Clients send requests to servers, specifying the type of service or data they require.
- Servers process these requests and send back responses containing the requested information or services.

2. Client-Server Communication

- Communication between clients and servers typically occurs over a network, such as the internet or a local area network (LAN).
- Communication protocols, such as HTTP, FTP, SMTP, and TCP/IP, govern the exchange of data between clients and servers.

3. Distributed Processing

- The client-server model enables distributed processing, where tasks are divided between clients and servers.
- Clients handle user interfaces, while servers manage data processing, storage, and application logic.

4. Scalability

- The client-server architecture allows for scalability by adding more servers or upgrading server hardware to handle increasing client demands.

5. Centralized Resources

- Servers centralize resources, making it efficient to manage and secure data, applications, and services.
- Clients access these centralized resources as needed.

6. Load Balancing

- In scenarios with high traffic, load balancers distribute client requests among multiple servers to ensure even load distribution and maintain performance.

Examples of the Client-Server Model

1. **Web Browsing** When you visit a website, your web browser (client) sends a request to a web server to retrieve web pages and resources (HTML, CSS, JavaScript) that are displayed to you.
2. **Email** Email clients (e.g., Outlook, Gmail) communicate with email servers (e.g., SMTP and IMAP servers) to send, receive, and store email messages.

3. **Database Management** Client applications connect to database servers (e.g., MySQL, Oracle) to retrieve, update, or manipulate data stored in databases.
4. **File Sharing** Clients request files or data from file servers (e.g., FTP servers) over a network.
5. **Video Streaming** Clients (media players or web browsers) connect to media servers to stream video content over the internet.

CLIENT-SERVER MODEL LAYERS

The client layer, business layer, and database layer are three critical components of a multi-tier architecture commonly used in web and application development. These layers work together to create robust and scalable software applications. Let's explore each layer in more detail

1. Client Layer

- The client layer is the topmost layer in the architecture, representing the user interface and user interaction with the application.
- It includes various client-side components such as web browsers, mobile apps, desktop applications, or any user-facing interfaces.
- Responsibilities of the client layer include presenting data to users, receiving user input, and rendering the application's graphical user interface (GUI).
- In web development, this layer is often built using technologies like HTML, CSS, and JavaScript. Mobile apps have their own client-side frameworks and libraries.
- The client layer communicates with the business layer to request data or trigger actions in response to user interactions.

2. Business Layer (Application Layer)

- The business layer is the central processing component of the application, responsible for implementing the application's logic, rules, and functionality.
- It acts as an intermediary between the client layer and the database layer, handling data processing, business logic, and application-specific operations.
- Key responsibilities include data validation, authentication, authorization, and orchestrating interactions with the database layer.
- The business layer ensures that data is processed correctly, business rules are enforced, and the application functions as intended.

- In some architectural patterns, the business layer is also known as the application layer or service layer.

3. Database Layer

- The database layer, often referred to as the data layer, is the lowest layer in the architecture, responsible for storing, retrieving, and managing data.
- It includes one or more database management systems (DBMS) or data storage solutions where data is stored in structured formats.
- Common types of databases used in this layer include relational databases (e.g., MySQL, PostgreSQL), NoSQL databases (e.g., MongoDB, Cassandra), or a combination of both.
- The database layer ensures data persistence and provides mechanisms for querying and modifying data.
- It responds to requests from the business layer to fetch or update information in the database.

Interactions Between Layers

- The client layer communicates with the business layer to request specific services or data.
- The business layer processes these requests, often involving business logic and validation.
- If data operations are required, the business layer interacts with the database layer to fetch or modify data.
- The database layer responds to database-related requests and sends data back to the business layer.
- Finally, the business layer delivers the processed data or response to the client layer for presentation to the user.

Benefits of Multi-Tier Architecture

- Scalability Each layer can be scaled independently to accommodate changes in user load or resource requirements.
- Separation of Concerns It enforces a clear separation of responsibilities between layers, making the application more maintainable and modular.
- Security Sensitive data and business logic are often concentrated in the business layer, enhancing security.

- Reusability Components in each layer can be reused in other parts of the application or in different applications.

1-TIER, 2-TIER 3-TIER ARCHITECTURES

1-Tier, 2-Tier, and 3-Tier architectures are different software architectural patterns used to design and structure applications based on the number of layers or tiers they employ. Each architecture has its own advantages and is suitable for different types of applications. Here's an explanation of each

1. 1-Tier Architecture

- 1-Tier architecture, also known as Single-Tier architecture, is the simplest form of application architecture.
- In a 1-Tier architecture, all the application components, including the user interface, business logic, and data storage, are tightly integrated into a single executable or program.
- This architecture is commonly found in small, standalone applications where there's no separation between the user interface and data processing logic.
- An example of a 1-Tier application is a simple calculator program on your computer.

Advantages of 1-Tier Architecture

- Simplicity It's straightforward to develop and manage due to its single-layer structure.
- Suitable for Small Apps Ideal for small, standalone applications with minimal complexity.

Disadvantages of 1-Tier Architecture

- Lack of Scalability Not suitable for applications that need to scale or grow.
- Limited Maintenance Hard to maintain and update as the application grows in complexity.
- Lack of Separation No clear separation of concerns, making it difficult to manage code and troubleshoot issues.

2. 2-Tier Architecture

- 2-Tier architecture, also known as Client-Server architecture, divides the application into two main components or tiers the client and the server.
- The client represents the user interface and application logic that runs on the user's device (e.g., desktop or mobile).

- The server handles data storage, processing, and business logic.
- Client and server communicate directly, often using protocols like SQL for database queries or HTTP for web applications.
- This architecture is commonly used in database applications and early web applications.

Advantages of 2-Tier Architecture

- Better Separation Improved separation of concerns compared to 1-Tier, making the application more maintainable.
- Scalable Suitable for applications with moderate complexity and scalability requirements.

Disadvantages of 2-Tier Architecture

- Limited Scalability Can become a bottleneck as the application grows due to the centralized server.
- Maintenance Challenges Still presents challenges in terms of code maintenance as the application scales.
- Limited Flexibility Changes to the client or server may impact each other.

3. 3-Tier Architecture

- 3-Tier architecture introduces an additional layer, separating the application into three tiers the presentation layer (client), the application logic layer (business logic), and the data storage layer (database).
- The presentation layer handles the user interface and user interactions.
- The application logic layer contains the business logic and processes user requests.
- The data storage layer manages data storage and retrieval.
- Communication between layers is usually well-defined and often relies on standard protocols or APIs.
- This architecture is widely used in modern web applications, where the client (browser) interacts with a web server, which in turn communicates with a database server.

Advantages of 3-Tier Architecture

- Modularity Clear separation of concerns and modular design, making it highly maintainable and scalable.
- Scalability Easier to scale each tier independently, allowing for more flexibility as the application grows.
- Improved Security Data storage and processing are isolated, enhancing security.

- Team Collaboration Facilitates collaboration among development teams working on different tiers.

Disadvantages of 3-Tier Architecture

- Complexity More complex than 2-Tier and may require additional development effort.
- Performance Overhead Communication between layers can introduce some performance overhead.

MODULE 1: RECAP FOR CSS

- Understanding the role of CSS in web development.
- The history and evolution of CSS.
- CSS syntax and basic structure Inline, internal, and external CSS

Understanding the Role of CSS in Web Development

- CSS (Cascading Style Sheets) is a fundamental technology for web development.
- It is used to control the presentation and styling of web pages.
- CSS separates the content (HTML) from its presentation, allowing for more flexibility and consistency in web design.
- It plays a crucial role in creating visually appealing and user-friendly websites.

The History and Evolution of CSS

- CSS was introduced in the late 1990s to address the limitations of HTML for styling web content.
- CSS1 was the first official CSS specification, followed by CSS2, CSS2.1, and CSS3.
- Each CSS version brought new features and capabilities, improving the design possibilities for web developers.
- CSS3 introduced many advanced features like animations, transitions, and responsive design.

CSS Syntax and Basic Structure

- CSS uses a simple and intuitive syntax to define styles for HTML elements.
- A CSS rule consists of a selector and a declaration block. For example

```
h1 {
  color blue;
  font-size 24px;
}
```

- The selector targets HTML elements you want to style (e.g., h1 selects all <h1> elements).
- The declaration block contains one or more property-value pairs that define the styling rules.
- Properties determine what aspect of the element you want to style (e.g., color for text color).
- Values specify how the property should be styled (e.g., blue for the color property).
- CSS rules are typically placed in a separate CSS file or included directly in an HTML document using <style> tags.

Inline, Internal, and External CSS

- CSS can be applied in different ways to HTML documents.
- Inline CSS Styles are applied directly to individual HTML elements using the style attribute. For example

** This is a green paragraph. **

Internal CSS Styles are defined within the <style> element in the HTML document's <head>. For example

```
<style>
  h1 {
    font-size 24px;
    color red;
  }
</style>
```

External CSS Styles are placed in a separate .css file and linked to the HTML document using the <link> element. For example

<link rel="stylesheet" type="text/css" href="styles.css">

MODULE 2: CSS SELECTORS AND PROPERTIES

- CSS Selectors and Properties
- Understanding CSS selectors (element, class, ID, universal, etc.)
- Combining selectors for more specific targeting
- Common CSS properties (color, font-size, margin, padding, etc.)
- CSS shorthand properties

Understanding CSS Selectors

- CSS selectors are patterns used to select and target HTML elements for styling.
- Common types of CSS selectors include
    - **Element selectors** Select elements based on their HTML tag name (e.g., h1, p, a).
    - **Class selectors** Select elements based on their class attribute (e.g., .button, .highlight).
    - **ID selectors** Select a single unique element based on its id attribute (e.g., #header, #main-content).
    - **Universal selector** Selects all elements on the page using *.

Combining Selectors for More Specific Targeting

- CSS allows you to combine multiple selectors to target specific elements precisely.
- You can use
    - **Descendant selectors** Target elements that are descendants of another element (e.g., ul li selects all list items within unordered lists).
    - **Child selectors** Select immediate children of an element (e.g., ul > li selects only direct children of unordered lists).
    - **Adjacent sibling selectors** Select elements that are adjacent siblings of another element (e.g., h2 + p selects paragraphs immediately following <h2> elements).
    - **General sibling selectors** Select elements that are siblings of another element (e.g., h2 ~ p selects all paragraphs that are siblings of <h2> elements).

Common CSS Properties

- CSS properties determine how selected elements are styled. Here are some common properties
    - **color** Sets the text color (e.g., color blue;).
    - **font-size** Controls the size of the font (e.g., font-size 16px;).
    - **margin** Defines the space around an element (e.g., margin 10px;).
    - **padding** Specifies the space inside an element (e.g., padding 5px;).
    - **background-color** Sets the background color of an element (e.g., background-color #f0f0f0;).
    - **border** Adds a border around an element (e.g., border 1px solid #ccc;).

- **text-align** Controls the alignment of text within an element (e.g., text-align center;).
- **display** Defines how an element is displayed (e.g., display block;, display inline;).

CSS Shorthand Properties

- CSS provides shorthand properties to set multiple related properties at once. For example
  - **margin** Combines margin-top, margin-right, margin-bottom, and margin-left.
  - **padding** Combines padding-top, padding-right, padding-bottom, and padding-left.
  - **border** Combines border-width, border-style, and border-color.

MODULE 3 CSS BOX MODEL

- Understanding the box model (content, padding, border, margin)
- Box sizing (content-box vs. border-box)
- Controlling element dimensions with the box model
- Margin collapsing

Understanding the Box Model

- The CSS Box Model is a fundamental concept in web design.
- It describes how elements on a web page are structured in terms of content, padding, border, and margin.
- Every HTML element can be thought of as a rectangular box.
- The key components of the box model are
  - **Content** The inner content of the element, where text or other HTML elements are displayed.
  - **Padding** The space between the content and the element's border.
  - **Border** A line that surrounds the padding and content.
  - **Margin** The space outside the element's border, which separates it from other elements.

Box Sizing (content-box vs. border-box)

- The default box-sizing property value is content-box.

- **content-box** The width and height of an element apply only to the content area. Padding, border, and margin are added to these dimensions.

- **border-box** The width and height of an element include the content, padding, and border. Margin is still added outside these dimensions.
- Using box-sizing border-box can simplify layout calculations.

Controlling Element Dimensions with the Box Model

- You can control an element's dimensions by setting its width and height properties.

- For example, to set the width of an element

```
.box {
  width 200px;
}
```

- Adding padding and border increases the total dimensions of the element.
- For example, with box-sizing content-box, if an element has a width of 200px and padding of 10px, the total width will be 220px (200px content + 10px padding on each side).

Margin Collapsing

- Margin collapsing is a behavior where the top and bottom margins of adjacent elements can overlap, resulting in a smaller margin.
- Margin collapsing typically occurs between sibling elements or between parent and child elements.
- To prevent margin collapsing, you can use techniques like adding padding or borders, using the overflow property, or setting display inline-block or display table.

MODULE 4 TEXT STYLING WITH CSS

- Fonts and text properties (font-family, font-size, font-weight, text-align, etc.)
- Styling links and text decoration
- Text transformation and spacing
- Web-safe fonts and custom font usage

Fonts and Text Properties

- CSS provides several properties for styling text and fonts to enhance the readability and aesthetics of web content.

- **font-family** This property specifies the font family for text, allowing you to choose from various typefaces. For example

body {

font-family Arial, Helvetica, sans-serif;

}

font-size It controls the size of the font. You can specify it in pixels, ems, or other units. For example

h1 {

font-size 24px;

}

- **font-weight** This property sets the thickness or boldness of the font. Common values are normal, bold, bolder, or numeric values like 400 or 700.
- **text-align** It determines how text is aligned within its container. Values include left, right, center, and justify.

Styling Links and Text Decoration

- Links are an essential part of web content, and CSS allows you to style them to make them stand out.
- **text-decoration** Use this property to control the underlining or decoration of links. For example, to remove underlines from links

a {

text-decoration none;

}

color You can change the color of links using the color property. For example

a {

color #007bff;

}

- Pseudo-classes like hover and visited can be used to define styles for different link states.

Text Transformation and Spacing

- **text-transform** This property allows you to transform text to uppercase, lowercase, or capitalize the first letter of each word.
- Example

```
p.uppercase {
  text-transform uppercase;
}
```

- **letter-spacing** Use this property to adjust the space between characters. Positive values increase spacing, while negative values decrease it.

Web-Safe Fonts and Custom Font Usage

- Web-safe fonts are fonts that are widely available on most devices and browsers.
- Examples of web-safe fonts include Arial, Helvetica, Times New Roman, and Georgia.
- You can specify a font stack, listing multiple fonts in font-family, to ensure graceful degradation if a preferred font is not available

```
body {
  font-family "Open Sans", Arial, sans-serif;
}
```

- Custom fonts can be loaded into web pages using the @font-face rule and font files (e.g., WOFF(Web Open Font Format WOFF) or TTF (True Type Face formats).
- Example of using a custom font

```
@font-face {
  font-family "CustomFont";
  src url("custom-font.woff") format("woff");
}

body {
  font-family "CustomFont", Arial, sans-serif;
}
```

MODULE 5 CSS LAYOUT

- Block-level vs. inline-level elements
- Controlling element positioning (position property)
- Display property (block, inline, inline-block, none, etc.)
- Floats and clearfix
- Flexbox layout

- CSS Grid layout

Block-level vs. Inline-level Elements

- In HTML, elements are categorized into two main types block-level and inline-level elements.
- **Block-level elements** These elements create a new block formatting context and typically start on a new line, occupying the full width of their parent container. Examples include , <p>, and <h1>.
- **Inline-level elements** These elements flow within the content and do not start on a new line. They occupy only as much width as necessary. Examples include , <a>, and .

Controlling Element Positioning (Position Property)

- The position property in CSS allows you to control the positioning of elements on a web page.
- Values for the position property include
    - **static (default)** Elements are positioned according to the normal flow of the document.
    - **relative** Elements are positioned relative to their normal position. You can use top, right, bottom, and left properties to adjust their position.
    - **absolute** Elements are positioned relative to their nearest positioned ancestor (an element with a position value other than static).
    - **fixed** Elements are positioned relative to the viewport, making them stay in the same place even when the page is scrolled.

Display Property (block, inline, inline-block, none, etc.)

- The display property defines how an element should be displayed on the web page.
- Common values include
    - **block** Elements generate a block-level box, forcing a line break before and after the element.
    - **inline** Elements generate an inline-level box and do not cause line breaks.
    - **inline-block** Elements generate an inline-level box but can have block-level properties and line breaks.

- **none** Elements become invisible and do not occupy space on the page. Useful for hiding elements dynamically.

Floats and Clearfix

- The float property allows elements to be pushed to one side of their containing element, causing other content to wrap around them.
- clear property is used to control how elements should behave concerning floated elements. It can take values like left, right, both, or none.

Flexbox Layout

- Flexbox (Flexible Box Layout) is a powerful layout model in CSS for creating complex layouts with ease.

- It introduces the display flex; property for creating a flex container and various properties like flex-direction, justify-content, and align-items for controlling the layout of flex items.
- Flexbox is excellent for creating responsive and evenly distributed layouts.

CSS Grid Layout

- CSS Grid Layout is another layout model that allows you to create grid-based layouts with rows and columns.

- It is especially useful for designing grid-like structures, such as magazine-style layouts, without needing complex HTML or CSS hacks.
- You can define rows and columns, place items in specific grid areas, and control the spacing between grid items.

MODULE 6 RESPONSIVE WEB DESIGN

- Introduction to responsive design
- Media queries and responsive breakpoints
- Creating fluid layouts
- Mobile-first vs. desktop-first approaches

Introduction to Responsive Design

- Responsive web design is an approach to web development that ensures web pages look and function well across various devices and screen sizes.

- It aims to provide an optimal user experience regardless of whether a user is accessing a website on a desktop computer, tablet, or smartphone.
- Responsive design achieves this by adapting the layout, content, and functionality of a website to different screen sizes and resolutions.

Media Queries and Responsive Breakpoints

- Media queries are a crucial aspect of responsive web design.
- A media query is a CSS technique that allows you to apply different styles based on various conditions, such as screen width, device orientation, or pixel density.
- Media queries use the @media rule to specify the conditions under which a set of CSS rules should apply.
- Example of a simple media query for screens narrower than 768 pixels

```
@media (max-width 768px) {
  /* CSS rules for smaller screens go here */
}
```

- Responsive breakpoints are specific points at which you adjust your design to fit the screen better. Common breakpoints include those for mobile (e.g., 320px), tablet (e.g., 768px), and desktop (e.g., 1024px) screens.

Creating Fluid Layouts

- Fluid layouts are layouts that adapt to the available screen space, making use of percentages rather than fixed pixel values for widths and heights.
- Fluid grids and flexible images are essential components of a responsive layout.
- CSS properties like max-width, width, and height are used to create fluid layouts that scale with the screen size.

Mobile-First vs. Desktop-First Approaches

- Mobile-first and desktop-first are two different approaches to responsive web design.
- **Mobile-first** In this approach, the default styles are designed for mobile devices with small screens. As the screen size increases (e.g., tablet and desktop), additional styles are added to enhance the layout.

- **Desktop-first** This approach starts with the default styles designed for larger screens (desktop). As the screen size decreases (e.g., tablet and mobile), styles are modified or removed to adapt to smaller screens.
- Mobile-first is recommended because it ensures a more efficient and performance-focused design, with extra features added for larger screens.