HTML5

HTML stands for Hyper Text Markup Language.

HTML is a Markup Language which is used to develop Static web pages.

We will apply styles to the web pages by using CSS( Cascading Style Sheet).

We will make dynamic web pages with the help of javaScript.

We will save the HTML files with “.htm” or “.html” extension.

We will execute the HTML files (web pages) with the help of browsers (chrome, opera, edge, Mozilla).

# Versions of HTML:

1. HTML 1.0
2. HTML 2.0
3. HTML 3.2
4. HTML 4.0
5. XHTML
6. HTML 5.0(Current Version)- Initial version released in 2008 – supports APIs

* Web Socket API
* Web Worker API
* Storages API
* Drag & Drop API
* Location API
* Canvas & SVG API
* Semantic Elements
* Cookies
* WebSQL

# Structure Of Webpage:

<!DOCTYPE HTML>

<HTML> 🡪ROOT ELEMENT

<HEAD>

HEAD SECTION

<META>

<TITLE></TITLE>

<STYLE></STYLE>

<LINK>

<SCRIPT></SCRIPT>

</HEAD>

<BODY>

BODY SECTION

</BODY>

</HTML>

DOCTYPE representing the version of HTML .

<html></html> tag representing the root element.

<head></head> tag, used to represent the header section.

<body></body> tag used to represent the body section.

# Semantic Tags

* HTML5 provides semantic elements
* Because of these elements readability of web pages increases.
* Semantic Tags:

1. <header>
2. <main>
3. <footer>
4. <aside>
5. <nav> --below Header
6. <article>
7. <audio>
8. <video>
9. <section> -- collaborates no of article and other tags .
10. <summary>
11. <details> --child tag for this is <summary>
12. <picture> --According to screen devices the different images are displayed.
13. <figure>
14. <figcaption>



# Storages API:

* HTML5 supports storages API.
* It supports two API’s in Storages

1. Local Storage—(items in Amazon Cart uses local storage)
2. Session Storage

* **Local Storage**
* Whenever we close the browser or open the new tab, we won’t loss the data.
* Local storage Capacity is 5MB per app per browser.
* **Session Storage**
* Whenever we close the browser or open the new tab, we will loss the data, then it is called session storage.
* Session Storage Capacity is based on System Memory.
* Javascript is a Object Based Scripting Language, it provides Window Object(Predefined Object).
* By using Window object only we access both local or session storage.
* How to access storages

Example: -

* window.localStorage
* Window.sessionStorage
* API functions:

1. setItem() – is used to store the data in the form of a “key&value” pairs.
2. getItem()- used to read the data.
3. removeItem() – used to delete the data based on “key”

* Example:

<script>

        function store(){

            /\*console.log(document.getElementById("storageemail").value,

            document.getElementById("storagepassword").value);\*/

            obj={

                "email":document.getElementById("storageemail").value,

                "password":document.getElementById("storagepassword").value

            }

            window.localStorage.setItem("login\_details",JSON.stringify(obj));

window.sessionStorage.setItem("login\_details",JSON.stringify(obj));

        }

        function read(){

            str=window.localStorage.getItem("login\_details");

let str2= window.sessionStorage.getItem("login\_details");

            obj1=JSON.parse(str);

            console.log(obj1);

        }

        function remove(){

            window.localStorage.removeItem("login\_details");

window.sessionStorage.removeItem("login\_details");

        }

    </script>

What is the difference b/w html and html5?

What is the semantic elements?

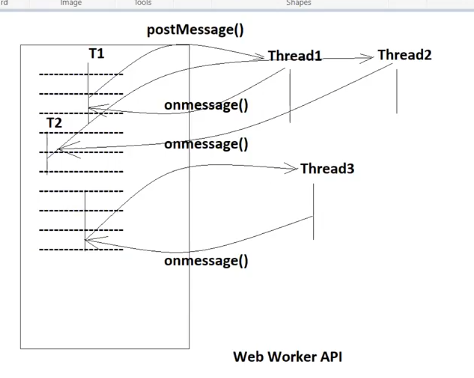
What are storages? –local and session

Are local Storage or Session Storage same?

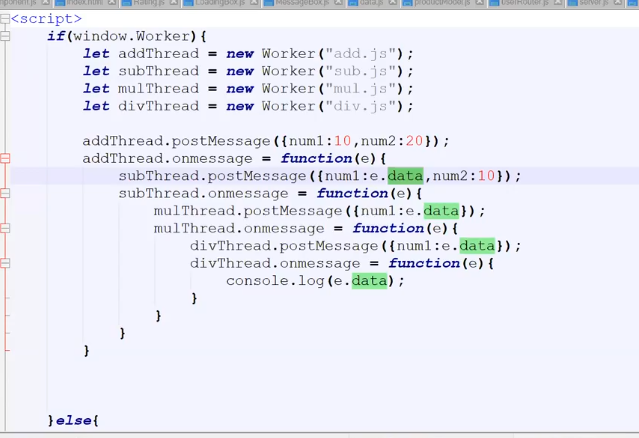
# Web Worker API

* Web Worker is used to create the multiprocesses.
* In Web Worker API ,there are two functions

1. postMessage() – To send the data to thread.
2. onmessage() – To get data from Thread.



Example:



Calculator.html

<script>

    //Whether the browser supports multithreading or not.

    if(window.Worker){

        //crating thread for executing add.js file

        let addThread=new Worker("add.js");

        //sending json object data to thread using postMessage method

        addThread.postMessage({num1:10,num2:20});

        //receving data from thread

        addThread.onmessage=function(e){

            console.log(e.data);

        }

    }

    else{

        console.log("Not supported");

    }

</script>

Add.js

this.onmessage=function(e){

    if(e.data!=undefined){

        this.postMessage(e.data.num1+e.data.num2);

    }

}

# Web Socket API

* Web Socket API is provided by HTML5.0
* Web Socket is used to build the Chat Applications ( Hand Shaking Mechanism Applications).
* The following are the steps to work with Web Socket API
* Create the http server
* Create the chat server
* Assign the port number to chat server with the http server
* Request for connection with chat server
* Read the client data
* Send the data to the another client
* To implement chat Application, we need to download one module /library
* >yarn add websocket

Or

* >npm install websocket
* Index.html
* <article id="msg"></article>
* <script>
* // Check if the browser supports WebSocket
* // WebSocket is not supported in older browsers, so we need this check
* if (window.WebSocket) {
* // Create a new WebSocket connection
* // The URL specifies the WebSocket server's location  ---ws specifies it is websocket communication
* let socket = new WebSocket("ws://localhost:8080/");
* // Event handler for when the connection is successfully opened
* socket.onopen = function () {
* // Send an initial message to the WebSocket server
* socket.send("Welcome to WebSocket");
* };
* // Event handler for when a message is received from the WebSocket server
* socket.onmessage = function (msgs) {
* // Append the received message (`msgs.data`) to the <article> element with id="msg"
* document.getElementById("msg").innerHTML += msgs.data + "<br>";
* };
* // Event handler for when an error occurs with the WebSocket connection
* socket.onerror = function (err) {
* // Log a generic error message to the browser console
* console.log("Error...!");
* };
* } else {
* // If WebSocket is not supported, notify the user
* console.log("WebSocket is not supported in this browser.");
* }
* </script>

Server.js

// Import HTTP module

// The HTTP module is a built-in Node.js module that is used to create an HTTP server

const http = require("http");

// Import WebSocket module

// The WebSocket module from the "websocket" package is used to create a WebSocket-based chat server

const WebSocketServer = require("websocket").server;

// Assign a custom port number to the chat server

// Create an HTTP server and make it listen on port 8080

// This HTTP server is used as a base for the WebSocket server

let socket = new WebSocketServer({

    httpServer: http.createServer().listen(8080, () => {

        console.log("Server listening on port 8080");

    }),

});

// Handle WebSocket connection requests, receive data, send responses, and close the connection

// `socket.on("request")` is triggered whenever a client sends a connection request

socket.on("request", function (request) {

    // Accept the connection request

    // The `request.accept` method creates a connection object for communication

    // `request.origin` is used to accept requests from any origin either http or https

    let connection = request.accept(null, request.origin);

    // Handle incoming messages from the client

    // Whenever the client sends data, the "message" event is triggered

    connection.on("message", function (message) {

        // Log the received message

        // The `utf8Data` property contains the message content in UTF-8 format

        console.log(message.utf8Data);

        // Send data back to the client

        // `sendUTF` is used to send UTF-8 formatted text messages

        connection.sendUTF("Hello\_1");

        connection.sendUTF("Hello\_2");

        // Send a delayed message to the client after 10 seconds

        setTimeout(() => {

            connection.send(new Date().toLocaleDateString());

        }, 10000);

    });

    // Handle connection closure

    // The "close" event is triggered when the connection is closed

    connection.on("close", function () {

        // Log a message indicating the connection is closed

        console.log("Connection closed");

    });

});

* To run the server , the following is used
* >node server
* Then run html file

# Location API

<article id="my\_location"></article>

<br><br>

<button onclick="clickMe()">Show Location</button>

<script>

    function clickMe() {

        // Check if the browser supports geolocation

        if (navigator.geolocation) {

            // Use the geolocation API to get the current position

            navigator.geolocation.getCurrentPosition(

                function success(position) {

                    // If successful, display the latitude and longitude in the <article> element

                    document.getElementById("my\_location").innerHTML =

                        "<b>Latitude    </b>"+position.coords.latitude + "<br><b>Lonitude   </b>" + position.coords.longitude;

                },

                function error(error) {

                    // Handle potential errors based on the error code

                    switch (error.code) {

                        case error.PERMISSION\_DENIED:

                            // User denied permission to access location

                            document.getElementById("my\_location").innerHTML = "PERMISSION DENIED";

                            break;

                        case error.POSITION\_UNAVAILABLE:

                            // Position information is unavailable

                            document.getElementById("my\_location").innerHTML = "POSITION UNAVAILABLE";

                            break;

                        case error.TIMEOUT:

                            // The request to get location timed out

                            document.getElementById("my\_location").innerHTML = "TIMEOUT";

                            break;

                        case error.UNKNOWN\_ERROR:

                            // An unknown error occurred

                            document.getElementById("my\_location").innerHTML = "UNKNOWN ERROR";

                            break;

                    }

                }

            );

        } else {

            // If geolocation is not supported by the browser, display an appropriate message

            document.getElementById("my\_location").innerHTML =

                "<b>Browser Not supporting Google Maps!! </b>";

        }

    }

</script>

* A collection of APIs to manage user navigation, device interactions, and browsing contexts.

### ****Core Components of Navigation API****

#### 1. ****Geolocation API****:

* **Purpose**: Retrieves the user's geographical location.
* **Methods**:
  + getCurrentPosition(success, error): Fetch current location.
  + watchPosition(success, error): Continuously monitor location.
  + clearWatch(id): Stops monitoring location.
* **Use Case**: Maps, location-based services, geotagging.

#### 2. ****History API****:

* **Purpose**: Manages browser session history.
* **Methods**:
  + pushState(state, title, url): Adds an entry to the history stack.
  + replaceState(state, title, url): Modifies the current history entry.
  + back(), forward(), go(n): Navigates in the history stack.
* **Use Case**: Single Page Applications (SPAs), seamless navigation.

#### 3. ****Navigator API****:

* **Purpose**: Provides information about the browser and device.
* **Key Properties**:
  + navigator.userAgent: Browser's user agent string.
  + navigator.language: User's preferred language.
  + navigator.onLine: Connectivity status.
* **Use Case**: Browser detection, feature compatibility checks.

#### 4. ****Navigation Timing API****:

* **Purpose**: Measures performance of navigation and page loading.
* **Key Properties**:
  + navigation.type: Navigation type (reload, back, forward).
  + timing.loadEventEnd: Page load completion time.
* **Use Case**: Performance monitoring, optimizing load times.

#### 5. ****Page Visibility API****:

* **Purpose**: Detects if the webpage is visible to the user.
* **Key Properties**:
  + document.visibilityState: Visibility state (visible or hidden).
* **Use Case**: Pause media when the user switches tabs.

#### 6. ****Network Information API****:

* **Purpose**: Provides details about the user's network.
* **Key Properties**:
  + navigator.connection.effectiveType: Network type (e.g., 4g, 3g).
* **Use Case**: Optimize resource usage for slower connections.

#### 7. ****Beacon API****:

* **Purpose**: Sends data to a server asynchronously.
* **Method**:
  + navigator.sendBeacon(url, data): Sends analytics/logs without affecting navigation.
* **Use Case**: Track events when users leave the page.

#### 8. ****Clipboard API****:

* **Purpose**: Manages clipboard operations like copy and paste.
* **Methods**:
  + navigator.clipboard.writeText(text): Copies text to clipboard.
  + navigator.clipboard.readText(): Reads text from clipboard.
* **Use Case**: Implement copy-to-clipboard functionality.

#### 9. ****Fullscreen API****:

* **Purpose**: Enables fullscreen mode for elements.
* **Methods**:
  + element.requestFullscreen(): Activates fullscreen.
  + document.exitFullscreen(): Exits fullscreen.
* **Use Case**: Enhanced viewing experiences for videos or games.

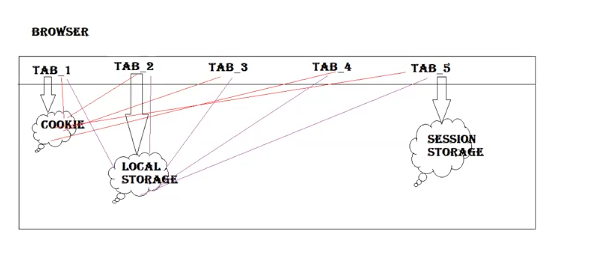
#### 10. ****Battery Status API****:

* **Purpose**: Provides information about battery status.
* **Key Properties**:
  + battery.level: Battery level as a percentage.
  + battery.charging: Indicates charging status.
* **Use Case**: Optimize behavior for low battery conditions.

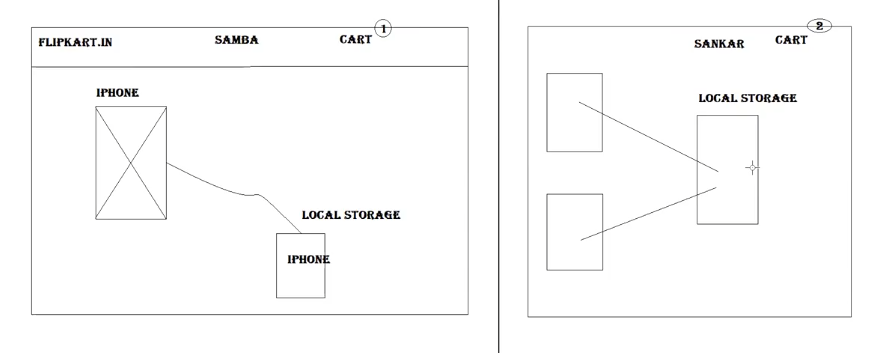
# Cookies

* Whenever we want to store less data then cookies are used.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Cookies** | **Local Storage** | **Session Storage** |
| Capacity | 4KB | 5MB / 10MB | Based on System configuration(RAM) |
| Browser’s Compactability | HTML 4.X / HTML 5.X | HTML 5.X | HTML 5.X |
| Access | Any Window | Any Window | Particular/Same Window |
| Expires | Manually Set(It won’t contain expiry date) | Never | On Window Close |
| Storage | Browser/Server | Browser | Browser |
| Request | Traverse from client to server | Never | Never |



* Local Storage, Session Storage and cookies are independent to user to user even on same browser.



Is it possible to set expiry date to cookies?

Are Cookies are safe? No

# Drag & Drop API

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Drag & Drop API</title>

    <style>

        article {

            width: 500px;

            height: 250px;

            border: 1px solid red;

        }

    </style>

    <script>

        function drag(event) {

            // Store the ID of the dragged element in the dataTransfer object

            event.dataTransfer.setData("key1", event.target.id);

        }

        function allowDrop(event) {

            // Allow dropping by preventing the default behavior

            event.preventDefault();

        }

        function drop(event) {

            event.preventDefault();

            // Retrieve the stored ID from the dataTransfer object

            let data = event.dataTransfer.getData("key1");

            // Append the dragged element to the drop target

            event.target.appendChild(document.getElementById(data));

        }

    </script>

</head>

<body>

    <!-- Drag-and-drop container -->

    <article id="article1" ondragover="allowDrop(event)" ondrop="drop(event)"></article>

    <br><br>

    <!-- Draggable image -->

    <img src="../images/shinchan.png"

         id="dragImage"

         width="200px"

         height="200px"

         alt="Image"

         draggable="true"

         ondragstart="drag(event)">

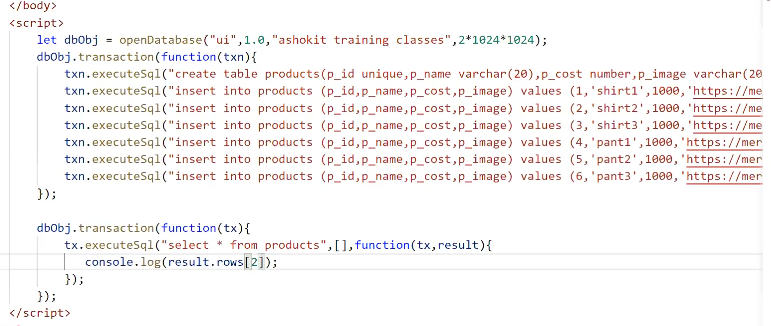
</body>

</html>

* First we need to set **draggable=”true”** for the object which we want to drag .
* Next we need to handle the event which is **ondragstart 🡪**what action should be performed when dragging starts**.**
* Next where we want to place ,their we need to handle the events such as **ondragover & ondrop .**

# WebSQL API

* Is WEBSQL is client side Database ? Yes
* Browser Side Database – At browser side if want to apply sql queries then WEBSQL is used.
* By using WebSQL API, we can apply SQL Queries at client side(browser side).
* Methods

1. openDatabase()
2. transaction()
3. executeSql()



# SVG API:

* Svg – Scalar vector graphics
* XML based
* SVG have more clarity while performing zoom operations
* SVG used to build the gaming application
* Drawbacks –SVG elements are heavy elements.

# Canvas API

# Forms

* Froms HTML5 onwards forms called ad “Web Forms 2.x”.
* Web Forms 2.x contains more features compared to html4.x forms
* Validation Support
* More types : 22types
* Markup element features enhancements
* **Input tag types**

|  |  |
| --- | --- |
| **HTML 4.X** | **HTML 5.X** |
| type=”email”   * Pattern * Size |  |
| type=”text” |  |
| type=”password” |  |
| type=”checkbox” |  |
| type=”radio” |  |
| type=”submit” |  |
| type==”button” |  |
| type==”file”   * accept |  |
| type=”image”   * src * alt |  |
| type=”hidden” |  |
| type=”reset” |  |
|  | type=”datetime”   * value * min * max |
|  | type=”datetime-local”   * value * min * max |
|  | type=”month” |
|  | type=”date”   * value * min * max |
|  | type=”week”   * value * min * max=”2018-w18” |
|  | type=”number” |
|  | type=”time” |
|  | type=”range”   * min attribute * max * step * value |
|  | type=”color”   * value attribute |
|  | type=”url”   * pattern * size |
|  | type=”search” |
|  | type=”tel” |

* **form tag Attributes:**
* method=”GET/POST/PUT”
* action=”/endpoint”
* oninput=”result.value=parseInt(a.value+parseInt(b.value)”
* **output Tag:**
* the output name is result of oninput tag.