Web Page

https://d2v4zi8pl64nxt.cloudfront.net/javascript-seo/5948abfc0e2df5.02876591.gif

Kameswari Chebrolu

Department of CSE, IIT Bombay

Outline

- **What constitutes a webpage?**
- What goes on inside a Browser?
  - What standard security mechanisms implemented?
- How do client and server communicate?
  - HTTP/HTTPs protocol
  - Session Management via cookies and tokens
- How does a web server process requests and generate responses?
  - Static vs Dynamic content

Web Page

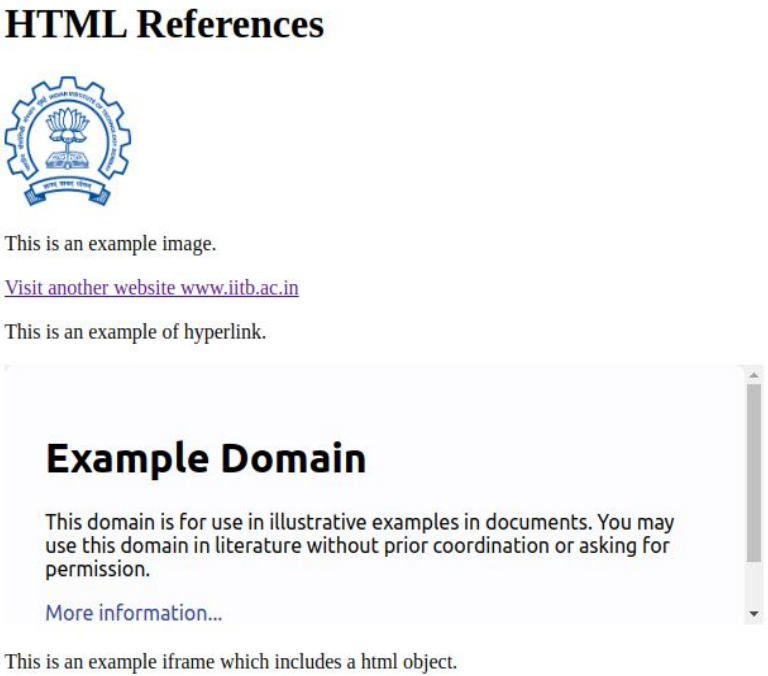
- Website made of Web Pages
- Web pages are written in HyperText Markup Language (HTML)
- **Web page** consists of base **HTML** file which includes several referenced **objects**
  - Object can be other HTML files, image files, Java applets, audio files etc
  - Text/Image that links to another page is called a **hyperlink** (often highlighted by some means)

- Each object is addressable by a **URL** (Uniform Resource Locator)
  - E.g. http://www.iitb.ac.in/images/header/iitb_logo.gif

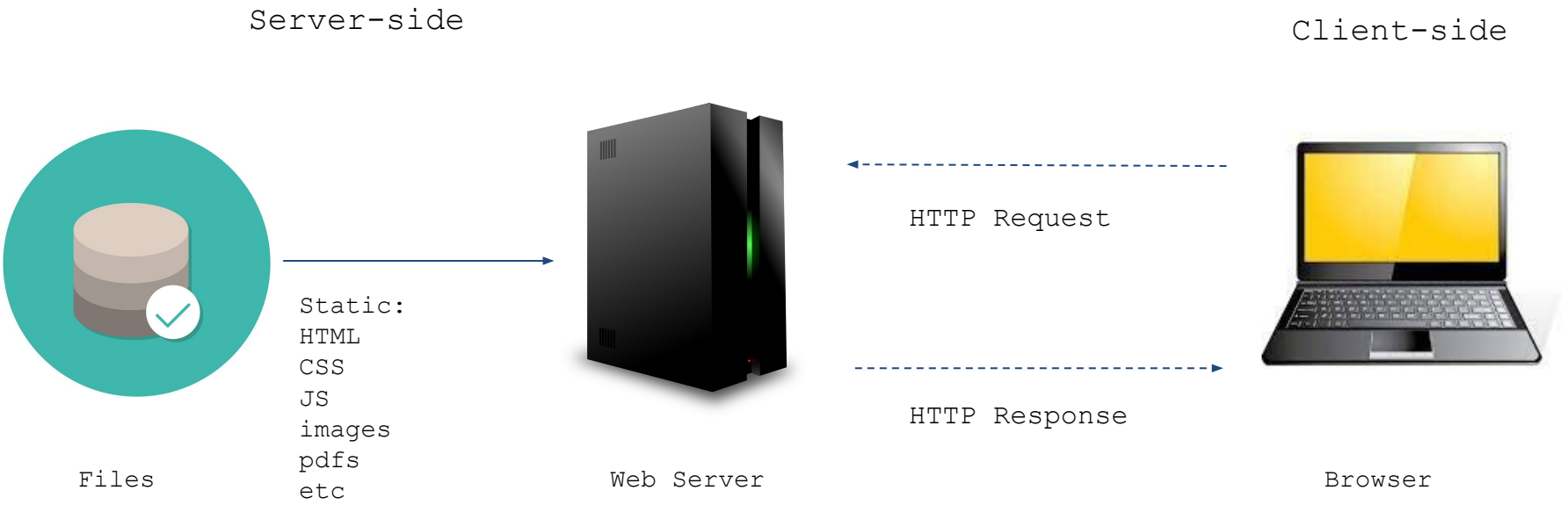
- Browser send requests for HTML and referenced objects and interpret received responses and displays content aesthetically

Sample HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Sample HTML</title>
</head>
<body>
        <h1>HTML References</h1>

        <!-- Image -->
        <img src="https://www.iitb.ac.in/sites/themes/touchm/logo.png"
alt="Example Image" width="100" height="100">
        <p>This is an example image.</p>

        <!-- Link -->
        <a href="https://www.iitb.ac.in/" target="_blank">Visit another website
www.iitb.ac.in</a>
        <p>This is an example of hyperlink.</p>

        <!-- Iframe -->
        <iframe src="https://example.com" width="600" height="200"
frameborder="0"></iframe>
        <p>This is an example iframe which includes a html object.</p>
</body>
</html>
```



- Early days of Internet, websites consisted mostly of static resources
  - Developers coded static (HTML) files by hand and deployed same on server
- User type website's URL in browser which results in request(s) to server (via HTTP protocol)
- Web server return static files (HTML) on disk in the form of HTTP response


Server-side
Client-side
Static:
HTML
CSS
JS
images
pdfs
etc
HTTP Request
HTTP Response
Files
Web Server
Browser

Cascading Style Sheets (CSS)

- HTML designed to define structure and semantics of a document, not so much presentation
  - Initially, developers used tags like <font>, <b>, and <i> to apply styles
  - Messy and hard-to-maintain code
    - Not easy to update/change visual design across pages without altering many files!

- Separate document content from presentation
  → led to development of CSS
  - Principle of separation of concerns
- Style sheet language used for describing the presentation of a document
  - Presentation: Layout, colors, fonts etc
  - Can also help create animations, responsive web designs
    - Responsive: adapt to different screen sizes and devices

HTML

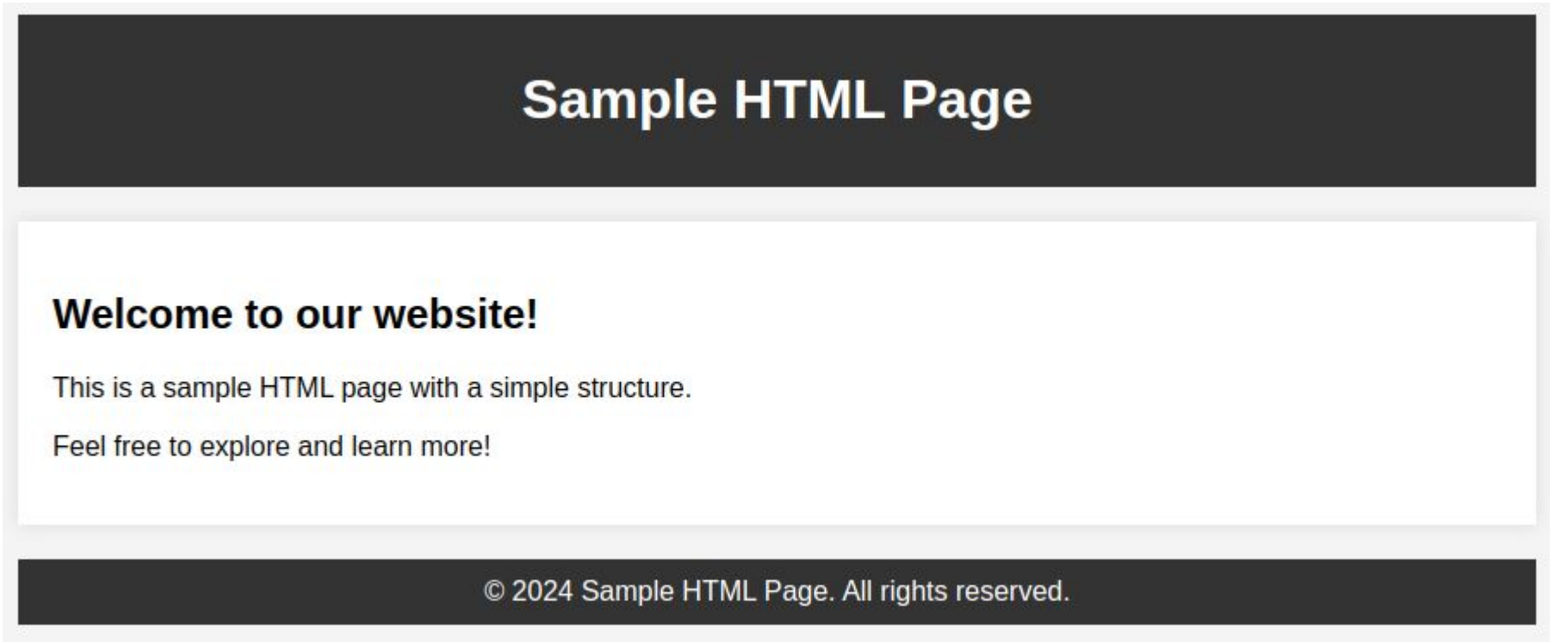
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sample HTML</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Sample HTML Page</h1>
    </header>

    <section>
        <h2>Welcome to our website!</h2>
        <p>This is a sample HTML page with a simple structure.</p>
        <p>Feel free to explore and learn more!</p>
    </section>

    <footer>
        &copy; 2024 Sample HTML Page. All rights reserved.
    </footer>
</body>
</html>
```

without styling

CSS

```css
/*styles.css*/
body {
  font-family: Arial, sans-serif;
  margin: 20px;
  padding: 0;
  background-color: #f4f4f4;
}
header {
  background-color: #333;
  color: #fff;
  padding: 10px;
  text-align: center;
}
section {
  margin: 20px 0;
  padding: 20px;
  background-color: #fff;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
footer {
        text-align: center;
  padding: 10px;
  background-color: #333;
  color: #fff;
}
```

Browser Output



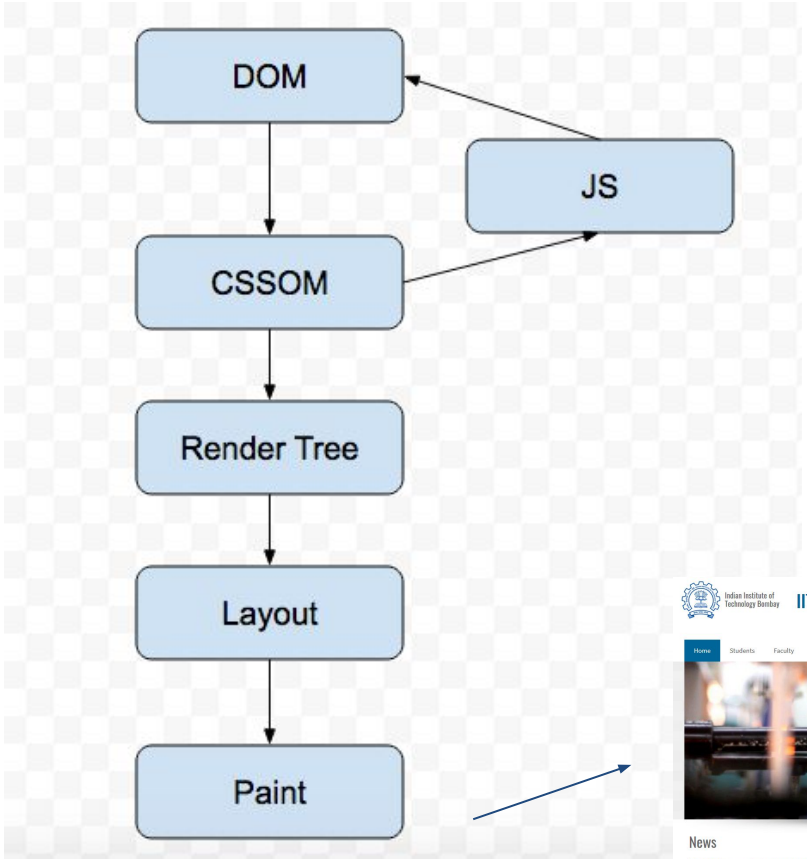
JavaScript

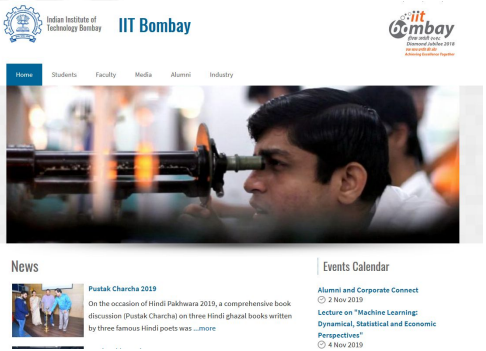
- A growing demand for more interactive and dynamic content → Client-Side Scripting
  - Originated in Netscape Communications, initially named Mocha and later LiveScript
  - Netscape and Sun later entered a collaboration, and renamed it Javascript
    - Why? Java of Sun Microsystems very popular, renamed to leverage popularity of Java
    - Note: JavaScript and Java are different languages with different purposes!

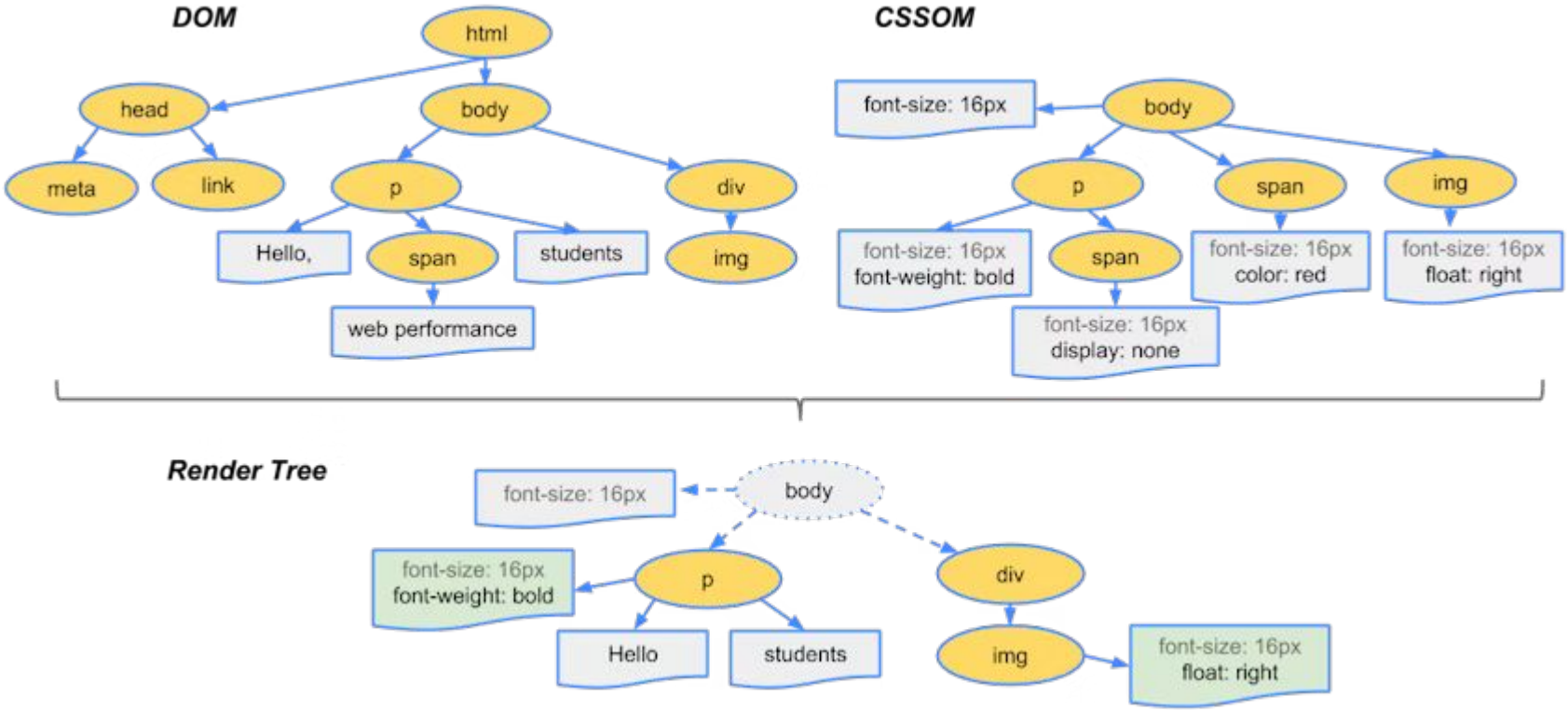
- Lots of nice features:
  - Light-weight
  - Cross-Platform Compatibility (across browsers, OS)
  - Can interact with Document Object Model (DOM)
    - Helps manipulate elements in the HTML document
- Standardized in 1997; led to widespread adoption

Document Object Model (DOM)

- An application programming interface (api) that extracts a tree structure out of HTML
  - Each node is an object representing a part of the document
  - Objects can be manipulated programmatically via JavaScript


DOM
JS
CSSOM
Render Tree
Layout
Paint


Indian Institute of Technology Bombay
IIT Bombay
Home
Students
Faculty
Media
Alumni
Industry
News
Pustak Charcha 2019
On the occasion of Hindi Pakhwara 2019, a comprehensive book discussion (Pustak Charcha) on three Hindi ghazal books written by three famous Hindi poets was ...more
Events Calendar
Alumni and Corporate Connect
2 Nov 2019
Lecture on "Machine Learning: Dynamical, Statistical and Economic Perspectives"
4 Nov 2019


DOM
html
head
body
meta
link
p
div
Hello,
span
students
img
web performance
CSSOM
font-size: 16px
body
p
span
img
font-size: 16px
font-weight: bold
span
font-size: 16px
color: red
font-size: 16px
float: right
font-size: 16px
display: none
Render Tree
font-size: 16px
body
font-size: 16px
font-weight: bold
p
div
Hello
students
img
font-size: 16px
float: right

DOM Manipulation

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

- In DOM, all HTML elements are defined as objects
- Example: document.getElementById("demo").innerHTML = "Hello World!";
  - Document is the object; getElementById is a method of document object, while innerHTML is a property

- Evolved new features
  - Libraries like jQuery
    - Simplifies DOM manipulation, event handling, animation, and Ajax interactions
  - Frameworks such as Angular, React, and Vue.js
    - Pre-written reusable code libraries or sets of tools
    - Help simplify and streamline development of web applications

A Few Details

- JavaScript has C-style syntax
  - The usual: variables, data types, operators, control structures (such as loops and conditionals), functions, and objects

```
let number = 42;
let text = "Hello";
let name = "Ravi";
let fruits = ['apple', 'banana', 'kiwi'];
console.log(text + name); // Outputs
Hello Ravi
```

```
// Example function
function add(a, b) {
    return a + b;
}

let result = add(3, 5);
console.log(result); // Outputs 8
```

DOM Manipulation

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>DOM Manipulation Example</title>
        <style>
                #myButton {padding: 10px;font-size: 16px;cursor: pointer;}
                #myDiv {margin-top: 20px;padding: 10px;border: 1px solid #ccc;}
        </style>
</head>
<body>
        <button id="myButton">Click me</button>
        <div id="myDiv">Initial content</div>

        <script>
        // Get references to the button and div elements
        var button = document.getElementById('myButton');
        var div = document.getElementById('myDiv');

        // Add a click event listener to the button
        button.addEventListener('click', function() {
        // Change the content and style of the div when the button is clicked
        div.innerHTML = 'Content updated!';
        div.style.backgroundColor = '#ffd700';  // Set background color to gold
        });
        </script>
</body>
</html>
```

Explanation

- HTML file includes a button (#myButton) and a div element (#myDiv)
- JavaScript code selects these elements using document.getElementById()
- Event listener is added to the button using addEventListener()

- When button clicked, provided callback function is executed
- Inside the callback function, the content of the div is updated using div.innerHTML
  - Style of div is also modified by changing its background color using div.style.backgroundColor

End Result: click button, content of div and background color changed

Asynchronous JavaScript and XML (AJAX)

- Allows web pages to communicate with a web server asynchronously
  - Can update parts of a web page without requiring a full page reload ⟶ faster response!
  - Modern AJAX implementations often use JSON (JavaScript Object Notation) instead of XML for data exchange
    - Why? JSON is lightweight and easy-to-parse

Core Components

- XMLHttpRequest Object:
  - While it says XML, supports JSON and plain text as well
- Fetch API: modern replacement for XMLHttpRequest
  - More flexible and powerful way to make HTTP requests
  - Better handling of errors, data types, chaining (create a sequence of asynchronous operations) etc

```html
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<title>XMLHttpRequest Example</title>
</head>
<body>
	<button onclick="fetchData()">Fetch Data</button>
	<div id="result"></div>

	<script>
	function fetchData() {
	// Create a new XMLHttpRequest object
	var xhr = new XMLHttpRequest();

	// Configure the request (GET method, URL)
	xhr.open('GET', 'https://example.com/todos/1', true);

	// Set up a callback function to handle the response
	xhr.onreadystatechange = function () {
	// Check if the request is complete (readyState 4) and successful (status 200)
	if (xhr.readyState === 4 && xhr.status === 200) {
		// Parse the JSON response
		var responseData = JSON.parse(xhr.responseText);

		// Update the DOM with the received data
		document.getElementById('result').innerText = `Title: ${responseData.title}`;
	}
	};

	// Send the request
	xhr.send();
	}
	</script>
</body>
</html>
```

XMLHttpRequest

Explanation

- A button (<button>) with an onclick attribute set to fetchData function
- A element with the ID set to result where the fetched data will be displayed
- fetchData function:
  - creates a new XMLHttpRequest object
  - configures it for a GET request to an API endpoint
  - sets up a callback function to handle the response
- onreadystatechange event checks if request is complete (readyState 4) and successful (status 200)
  - When met, JSON response is parsed, DOM updated with received data

Result: user clicks "Fetch Data" button, the fetchData function is called and results displayed

```html
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<title>Fetch API Example</title>
</head>
<body>
	<button onclick="fetchData()">Fetch Data</button>
	<div id="result"></div>

	<script>
	function fetchData() {
	// Using the Fetch API to make a GET request
	fetch('https://example.com/todos/1')
	.then(response => {
		// Check if the response status is OK (200-299)
		if (!response.ok) {
		throw new Error(`HTTP error! Status: ${response.status}`);
		}
		// Parse the JSON data from the response
		return response.json();
	})
	.then(data => {
		// Handle the data
		document.getElementById('result').innerText = JSON.stringify(data, null, 2);
	})
	.catch(error => {
		// Handle errors
		console.error('Fetch error:', error);
	});
	}
	</script>
</body>
</html>
```

fetch

Explanation

- A button (#fetchButton) to click and a div element (#result) to display the fetched data.
- JavaScript code adds event listener to the button
- When button is clicked, fetchData function is called
- fetchData function uses Fetch API to make a GET request to API endpoint

- fetch function returns a Promise that resolves to the Response object representing the response to the request
  - check if the response is okay (status code in the range 200-299).
  - If okay, use json method to parse the JSON data
- Parsed data is then displayed in the #result div
- Any errors caught and logged to console

References

- HTML: https://www.w3schools.com/html/
- CSS: https://www.w3schools.com/css/
- Javascript: https://www.w3schools.com/js/
- AJAX: https://www.w3schools.com/js/js_ajax_intro.asp and https://www.w3schools.com/js/js_api_fetch.asp

Summary

- Web Pages are written in HTML
- CSS specifies the presentation and styling of the HTML (XML) document
- Javascript makes web pages interactive
  - AJAX helps update web pages asynchronously