# **JavaScript Introduction**

## JavaScript Frameworks

Compatible with JavaScript

* AureliaJS,
* PolymerJS,
* Meteor - uses NodeJS
* Ember
* NodeJS
* BackboneJS
* Jquery
* VueJS
* ReactJS
* AngularJS

## JavaScript Libraries

Compatible with JavaScript

* Sweedler
* Sound effect
* Cliff chairs
* Cleave.js
* Html to canvas – pop motion
* Type.js
* Typed.js
* ScrollMagic
* Handson table
* Particles.js
* Dragula.js
* Rough.js
* QuillJS
* Materialize

# JavaScript Programming Basics

## Variables

## Operators

## Strings

## \*Regular Expressions

A regular expression (regex) is a character sequence used as a search pattern for text searches and replacements. It describes what you're searching for, from single characters to complex patterns. Regular expressions are versatile tools for various text operations.

Example:

Pattern: /w3schools/i

"w3schools" is the search pattern.

"i" is a modifier for case-insensitive search.

## Assignments

## Data Structures

## Functions

## Calling Functions within Functions

## Booleans

## Conditional Statements

## Creating Objects with user-defined functions

## Comparitors

# Document Object Model

## What is DOM?

Is an application programming interface (API) for HTML and XML documents and to access an object, property, or method, its reference must include every object that contains it, separated by a dot.

The DOM is what connects web pages to scripts or languages.

## JavaScript Objects

Window, Document (web page)

## Property

Property is regarded as characteristics of an object. With JavaScript on the other hand. The properties of objects can also be an object.

**The rule for dual identities is this:** a property is also an object if that property has its own properties and methods.

## Methods

Methods these are actions applied to objects, it’s what makes HTML documents react to the user.

Parameters are needed by a method to accomplish its task. An alert () method does nothing without the parentheses inside.

Method examples

* alert (“dialog box”)
* write (writes content to the page)
* focus (causes mouse cursor to be inserted into a form element)

## Core APIs in DOM

What is the core api?

Document and window objects are commonly used in DOM programming. Window represents something within the browser. And document is the root of document itself.

Element inherits from the generic Node interface.?

APIs in web & XML page scripting using the DOM

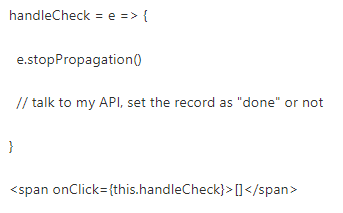
* document.getElementById(id)
* document.getElementsByTagName(name)
* document.createElement(name)
* parentNode.appendChild(node)
* element.innerHTML
* element.style.left
* element.setAttribute
* element.getAttribute
* element.addEventListener
* window.\_content
* window.onload
* window.dump()
* window.scrollTo()

## Event Bubbling

What is event Bubbling?

Event bubbling is an event that starts where its triggered. Then moves up to each parent until it reaches document.

1. Button clicked.
2. Element div
3. Element body
4. Element html
5. Document



Event Propagation

Stop propagation stops the bubbling effect from occurring to the parents.

Event delegation

Making parent an element as a listener for all events to happen inside it. Instead of adding event listeners for each. Add one to the parent element and target child.

# Events

## What are Events?

Events are user interactions or browser-related occurrences that can trigger JavaScript code execution.

## Event Types Pt1

Basic Events

* Onclick
* OnSubmit
* Onevent

## Event Types Pt2

Intermediate Events

* Focus (input focused)
* Blur (input lost focus)
* Change (value changed)
* Submit (form submitted)
* Load (page loaded)
* Unload (leaving the page)
* Resize (window resized)

## Event Types Pt3

Form Events

Events like submit, reset, focus, and blur can be used to enhance form interactions.

Document Events

Events related to the document include load, DOMContentLoaded, and unload.

Window Events

Window-related events include resize, scroll, and beforeunload.

Event Delegation

Attach event listeners to parent elements for better performance in handling multiple child elements.

## Manipulate content of webpage

Editing the style of an element

**id.style.backgroundColor = "blue";**

Editing the style of an ID

**class.textContent**

Preventing default behaviour of an event

**event.preventDefault();**

Manipulating Content

**var variableName = document.getElementById("target");**

**targetElement.innerHTML = “Button Clicked!”**

## Handling events

Using an event handler in JavaScript

**const var = document.getElementById("btn")**

Using event directly on element

**var.onclick = function() {**

**alert("Button clicked!");**

**};**

Using an event handler in HTML

**function handleClick() {**

**alert("Button clicked!");**

**}**

Event listener

**variable.addEventListener('change', function() {**

**alert ("Value changed!");**

**});**

# **JavaScript Intermediate Level**

## Frames and windows (popups)

Popups

Showing content from another window without closing the main window. OAuth authorization (login with google or Facebook) still makes use of this.

window.open(url)

Window.open

Syntax to open a popup:

window.open(url, name, params)

params (no spaces) can include:

* left/top (numeric)
* width/height (numeric)

Popup Blocking

Some websites abuse popups to take people outside their site. Secure sites make use of block popups that block popups that was not user triggered like onclick.

// popup blocked

**window.open('https://javascript.info');**

// popup allowed

**button.onclick = () => {window.open('https://javascript.info');}**

Setting Timeout

**setTimeout(() =>**

**window.open('http://google.com'), 3000);**

**setTimeout(() =>**

**window.open('http://google.com'), 1000);**

The result of running this code is that, after waiting for 3 seconds, a new browser window or tab will be opened, directing the user to the Google website.

The faster it open **the less chance it'll be blocked**

Window features

* menubar (yes/no) – shows or hides the browser menu on the new window.
* toolbar (yes/no) – shows or hides the browser navigation bar (back, forward, reload etc) on the new window.
* location (yes/no) – shows or hides the URL field in the new window. FF and IE don’t allow to hide it by default.
* status (yes/no) – shows or hides the status bar. Again, most browsers force it to show.
* resizable (yes/no) – allows to disable the resize for the new window. Not recommended.
* scrollbars (yes/no) – allows to disable the scrollbars for the new window. Not recommended.

<https://developer.mozilla.org/en-US/docs/Web/API/Window/open>

Minimalistic Window

Browser uses default parameters if third argument is not used.

Be sure to specify params or set features to yes or it’ll be under assumption that features have no value.

Window will open near last window opened if top/left params not specified.

Window would be same size as last window if no width/height is specified.

// Open a new window with default parameters

**let newWindow = window.open('/');**

// Open a new window with specified features

**let newWindowWithFeatures =**

**window.open('/', 'example', 'width=300,height=300');**

// Accessing popup from window

**newWindowWithFeatures.onload = function() {**

**newWindowWithFeatures.close();**

**alert(newWindowWithFeatures.closed);**

// true

**}**

Accessing window from popup

A popup may access the “opener” window as well using window.opener reference. It is null for all windows except popups.

// Accessing window from popup

**let popup =**

**window.open('/', 'example', 'width=300,height=300');**

**if (popup.opener) {**

**popup.opener.alert('Hello from the popup!');**

**}**

Closing a popup

One can use **window.close()** but is ignored if **window.open()** was not used.

// Closing a popup

**popup.close();**

Scrolling and resizing

Methods to move/resize window.

* **win.moveBy(x, y)**
* **win.moveTo(x, y)**
* **win.resizeBy(width, height)**
* **win.resizeTo(width, height)**

The browser blocks all these methods unless its popups that’s been opened with no additional tabs.

No minification/maximization

Move/resize methods don’t work for maximized/minimized windows.

Scrolling a window

**win.scrollBy(x, y)**

**win.scrollTo(x, y)**

**elem.scrollIntoView(top=true)**

// Scrolling and resizing

**newWindowWithFeatures.moveBy(100, 100);**

**newWindowWithFeatures.resizeBy(200, 200);**

// Scrolling a window

**newWindowWithFeatures.scrollBy(0, 100);**

Focus/Blur on a window

There are two ways to focus/unfocus on a window. The window.focus() and window.blur()

Malicious websites attempt to keep users on focused on the page:

window.onblur = () => window.focus();

some browsers try to protect its users and block the code out completely.

Things can be done to alleviate this issue:

* Use **newWindow.focus()**
* We can track window.onfocus/onBlur ensuring that the user is in the new window.

// Focus/Blur on a window

**newWindowWithFeatures.focus();**

**newWindowWithFeatures.onblur = () => newWindowWithFeatures.focus();**

## Cross-Window Communication

Same Origin Policy

* Limits access of windows and frames to each other.
* Ensures that different windows from different origins can't access each other's content.
* Origins are determined by the combination of protocol, domain, and port.

Same

http://site.com

http://site.com/

http://site.com/my/page.html

Not Same

http://www.site.com

http://site.org

https://site.com

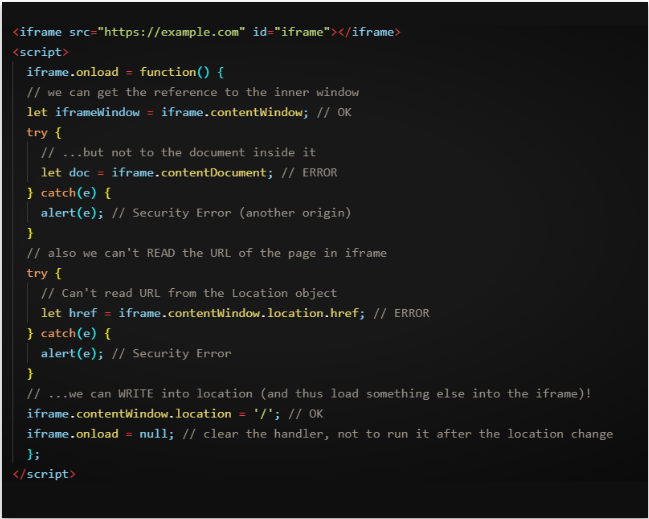
http://site.com:8080

iFrame

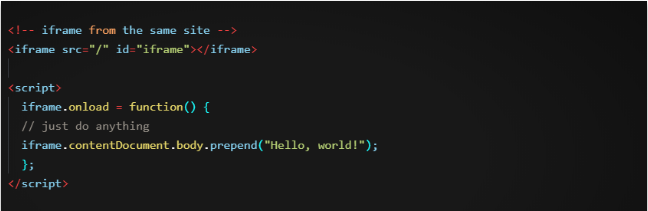
Embeds a window within a separate document and window.

Access using:

* iframe.contentWindow to get the window inside the <iframe>
* iframe.contentDocument to get the document inside the <iframe>, iframe.contentWindow.document.

Browser checks if the iFrame has the same origin before granting access.

iframe has same origin we can do anything with it.



iframe.onload vs iframe.contentWindow.onload

they the same which triggers when embedded windows loads with all resources.

We can’t access iframe.contentWindow.onload from another origin.

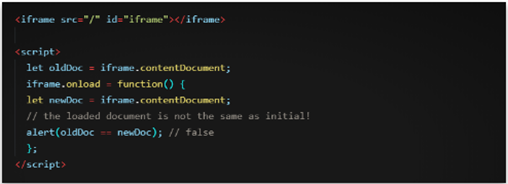
Windows on same subdomain: document.domain

John.site.com has same second-level domain name as peter.site.com

Can be treated as the same origin for cross window communication. Only possible for sits with same second-level domain.

document.domain = ‘site.com’

**Pitfall:** The iframe has a document immediately upon creation, but it's the wrong one. The correct one is loaded later.



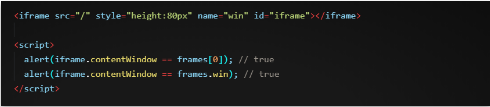
The right document is at the place when iframe.onload triggers. But it only triggers when the whole iframe with all resources is loaded.

Catch the moment earlier using checks in setInterval.



Collection: window.frames

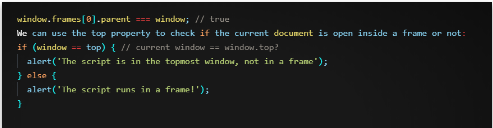
Getting window.objects for <iFrames> Make use of collectionwindow.frames:

* By number: window.frames[0] – the window object for the first frame in the document.
* By name: window.frames.iframeName – the window object for the frame withname="iframeName"

iFrames may have iFrames inside, creating hierarchy.

Navigation links are:

* window.frames – the collection of “children” windows (for nested frames).
* window.parent – the reference to the “parent” (outer) window.
* window.top – the reference to the topmost parent window.



Sandbox iFrame attribute

Sandbox iFrame attribute allows for exclusion of specific actions within an iFrame. Preventing execution from untrusted code.

Empty Default (restricted):

<iframe sandbox src="...">

Relaxed by including space-separated list of restrictions (less restricted):

<iframe sandbox="allow-forms allow-popups">

Limitations

allow-same-origin

allow-top-navigation

allow-forms

allow-scripts

allow-popups

Cross Window Messaging

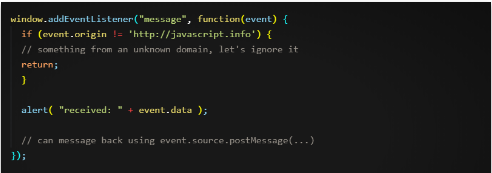
PostMessage interface allows windows to talk with each other regardless of the origin on condition that both sides accept the request.

Has 2 parts

* PostMessage: the window wanting to send a message, calls postMethod of the receiving window.
* Arguments: data that’s being sent (JSON.stringify), and the targetOrigin. This specifies origin of target window.

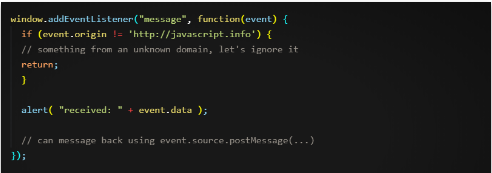
Target Origins ensures the window only receives data if it’s still at the correct site.



On Message

When receiving message on target window, target window makes use of a handler, and it triggers when postMethod is called.

Event object includes: Data, Origin, Source



## ClickJacking Attack

What is it?

Clickjacking allows malicious webpages to click on a victim site on behalf of the visitor.

ClickJacking is for clicks and taps (mobile), not keyboard.

Not visible to user because iframes are invisible.

**How it occurs:**

* Visitor is lured to the malicious site
* Page has catchy/harmless link (get rich now)
* Over the link resides a iframe with src from another site. For example FaceBook’s like button layered over it using z-index

<https://javascript.info/clickjacking#old-school-defences-weak>

Weak defences against clickjacking

Framebusting: forbids opening a page in a frame.

if (top != window) {

top.location = window.location;}

**Downsides:** window make itself the top if it finds out its on top. Theres many ways to hack around it.

Blocking top-navigation

Blocking the transition to top by changing **top.location** in **beforeunload** event handler

window.onbeforeunload = function() {

return false;}

The user would receive a message if they want to leave when the iFrame tries to change **top.location**

Sandbox attribute

<iframe sandbox="allow-scripts allow-forms" src="facebook.html"></iframe>

<https://javascript.info/clickjacking#old-school-defences-weak>

X-Frames Options

Server-Side Header X-Frame-Options

**Purpose:** It controls how a webpage is displayed when it's embedded within an iframe on another site. This header is set on the server and is sent as part of the HTTP response.

Values:

* **DENY:** The page cannot be displayed in a frame, regardless of where the request originates.
* **SAMEORIGIN:** The page can be displayed in a frame, but only if the parent document is from the same origin (i.e., the same domain).
* **ALLOW-FROM DOMAIN:** The page can be displayed in a frame, but only if the parent document is from the specified domain.

Twitter Example:

Twitter uses X-Frame-Options: SAMEORIGIN.

If you try to embed a Twitter page in an iframe, some browsers might display it as empty or show a message indicating that the browser won't allow it.

<iframe src="https://twitter.com"></iframe>

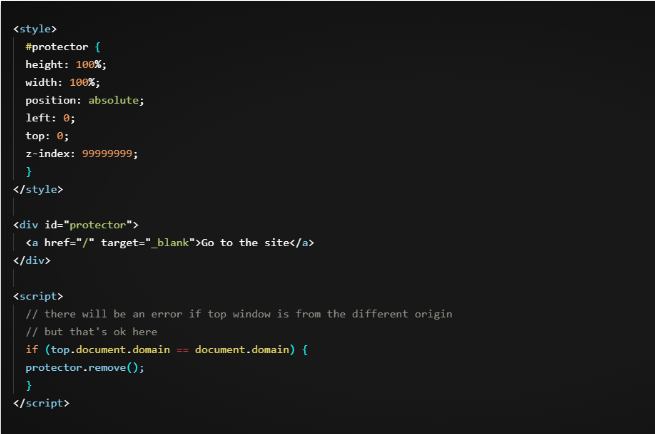
Considerations:

The behavior might vary depending on the browser.

Handling Disabled Functionality:

Due to X-Frame-Options, other sites might not be able to show our page in a frame.

As a workaround, you can cover the page with a <div> element styled to cover the entire viewport (100% width and height). This will accept all clicks, effectively blocking interaction with the underlying content.



<https://javascript.info/clickjacking#old-school-defences-weak>

Samesite cookie attribute

A cookie that prevents clickjacking attacks.

Prevents it by ensuring that we only sent to a website if its opened directly. Not iFrame.

<https://javascript.info/cookie#samesite>

*Set-Cookie: authorization=secret; samesite*

**Purpose:** The SameSite attribute is used to prevent certain types of security attacks, particularly clickjacking attacks. It determines whether cookies should be sent with cross-origin requests.

How It Works: SameSite prevents a cookie from being sent in the header of a cross-origin request. This means that the cookie will only be sent if the user is directly interacting with the site, not if the site is embedded in an iframe on a different domain.

Values:

Strict: The cookie will only be sent with requests originating from the same site the cookie was set on.

Lax: The cookie is sent with top-level navigations and will be sent along with GET requests initiated by third-party websites through an <img> tag, <link> tag, or <script> tag.

None: The cookie will be sent with every request, regardless of the source.

Example:

Set-Cookie: authorization=secret; samesite=Strict: This sets a cookie named authorization with the value secret and the SameSite attribute set to Strict.

Usage:

By setting the SameSite attribute appropriately, you can mitigate risks associated with certain types of attacks, especially clickjacking.

Note:

It's important to be cautious when using SameSite=None as it requires the cookie to be secure (i.e., transmitted over HTTPS). This value is commonly used for cookies that are intended for use by third-party sites.

## Array Buffer and Binary Arrays

ArrayBuffer

* We encounter binary data in file operations (create, download) and image processing.
* Not an array, has fixed length and memory space.
* Individual bytes accessed with views, not buffer[index].
* Raw bytes stored.
* Classes: ArrayBuffer, Uint8Array, Blob, File.

let buffer = new ArrayBuffer(16);

alert(buffer.byteLength); // 16

Manipulating ArrayBuffer

* Views (TypedArray) interpret data.
* Uint8Array treats each byte as a number (0 to 255).
* Uint16Array treats every 2 bytes as an integer (0 to 65535).
* Uint32Array treats every 4 bytes as an integer (0 to 4294967295).
* Uint64Array treats every 8 bytes as a floating point number.

let buffer = new ArrayBuffer(16);

// treat buffer as a sequence of 32-bit integers

let view = new Uint32Array(buffer);

// 4 bytes per integer

alert(Uint32Array.BYTES\_PER\_ELEMENT);

// 4, it stores that many integers

alert(view.length);

// 16, the size in bytes

alert(view.byteLength);

// let's write a value

view[0] = 123456;

// iterate over values

for(let num of view) {

// 123456, then 0, 0, 0 (4 values total)

alert(num);}

Typed Array

* Efficiently handle binary data.
* Similar to arrays, but with specific types.
* **Properties:** buffer, byteLength.
* Useful for large numeric data tasks.

Examples of TypedArray Constructors

**TypedArray(buffer, [byteOffset], [length]):**

* Uses existing ArrayBuffer as data source.
* Parameters: buffer, byteOffset (default 0), length.

let buffer = new ArrayBuffer(16);

let arr = new Uint8Array(buffer, 4, 6);

**TypedArray(object):**

* Creates TypedArray from array-like object.

let arr = new Uint8Array([0, 1, 2, 3]);

**TypedArray(typedArray):**

* Uses values from another TypedArray.

// Converts and copies values

let arr16 = new Uint16Array([1, 1000]);

let arr8 = new Uint8Array(arr16);

**TypedArray(length):**

* Creates TypedArray with specified elements.

// Creates an array with 4 elements

let arr = new Uint16Array(4);

**TypedArray():**

* Creates empty TypedArray.

// Creates an empty array

let arr = new Uint8Array();

Typed Arrays and Data Types

1. Uint8Array, Uint16Array, Uint32Array: For unsigned integers.
2. Uint8ClampedArray: Clamps values on assignment.
3. Int8Array, Int16Array, Int32Array: For signed integers.
4. Float32Array, Float64Array: For floating-point numbers.

<https://www.youtube.com/watch?v=UYkJaW3pmj0>

Out of Bounds Behavior

No error on out-of-range value. Value is truncated, keeping least significant bits.

For example, let's consider a Uint8Array, which can represent integers from 0 to 255 (8 bits).

**Case 1:** Storing 256 (Binary: 100000000, 9 bits)

The Uint8Array only has 8 bits available. The rightmost 8 bits are stored, resulting in 0 being saved. 0 to 255

let arr = new Uint8Array(1);

arr[0] = 256;

console.log(arr[0]); // Output: 0

**Case 2:** Storing 257 (Binary: 100000001, 9 bits)

Again, only the rightmost 8 bits are stored. This results in 1 being saved.

let arr = new Uint8Array(1);

arr[0] = 257;

console.log(arr[0]); // Output: 1

This behavior is due to the nature of binary representation. When a number is too large for the available bits, JavaScript essentially performs a modulo operation with a modulus of 2^8 (for Uint8Array).

Now, Uint8ClampedArray behaves differently. It's designed for image processing, where it's useful to automatically clamp values to the valid range (0-255).

If you attempt to store a value greater than 255, it will be clamped to 255.

If you attempt to store a negative value, it will be clamped to 0.

This is helpful in scenarios where you want to ensure that pixel values in an image stay within a valid range.

DataView

Versatile view over ArrayBuffer.

Allows access at any offset and format.

Dynamic format selection.

DataView Constructor

new DataView(buffer, [byteOffset], [byteLength]).

buffer: Underlying ArrayBuffer.

byteOffset: Starting byte position (default 0).

byteLength: Length in bytes (default till end of buffer).

let buffer = new ArrayBuffer(6);

let view = new DataView(buffer);

view.setUint8(0, 42);

view.setFloat32(2, Math.PI);

let num1 = view.getUint8(0);

let num2 = view.getFloat32(2);

Use Cases

ArrayBuffer: Binary data handling (files, image processing).

TypedArray: Efficient binary data processing.

DataView: Handling mixed-format data in a single buffer.

# **JavaScript Intermediate Level Quizzes**

## Quiz 1

What is the purpose of a popup in web development?

a) To display images

b) To show alerts

c) To open content in a new window without closing the main window

d) To play videos

Provide the syntax for opening a new window using the window.open() method, including its possible parameters.

a) window.open(url)

b) window.open(url, name, params)

c) window.open(url, params, name)

d) open.newWindow(url, params, name)

How can you ensure that a popup won't be blocked by the browser?

a) By using window.create()

b) By setting a timeout for opening the window

c) By only triggering popups on user actions like clicks

d) Popups cannot be prevented from blocking

What does the window.opener reference allow a popup window to do?

a) Access the parent window that opened it

b) Close the parent window

c) Open a new popup

d) Access a sibling window

How can you close a popup window programmatically?

a) window.closePopup()

b) popup.close()

c) window.close()

d) closeWindow(popup)

Name two methods to move or resize a window.

a) moveWindow(x, y) and resizeWindow(width, height)

b) win.moveBy(x, y) and win.resizeBy(width, height)

c) window.changePosition(x, y) and window.changeSize(width, height)

d) moveTo(x, y) and resizeTo(width, height)

Can you perform move/resize operations on maximized or minimized windows?

a) Yes, it works without any issues

b) No, these methods don't work on maximized/minimized windows

c) Only resizing is possible on maximized windows

d) Only moving is possible on minimized windows

How do you scroll a window using JavaScript?

a) window.scroll(x, y)

b) window.scrollTo(x, y)

c) elem.scrollIntoView(top=true)

d) All of the above

Explain the purpose of window.focus() and window.blur(). Why might they be used?

a) They control the visibility of windows

b) They control the focus of user interaction on windows

c) They control the background color of windows

d) They control the size of windows

How do you prevent malicious websites from keeping users focused on their page?

a) Use window.onblur = () => window.focus();

b) Close the window using window.close()

c) Use window.preventFocus()

d) It's not possible to prevent this behavior

## Quiz 2

What does the Same Origin Policy in web browsers primarily aim to do?

a) Enable access between any windows or frames.

b) Restrict access of windows or frames from different origins.

c) Control the display of popups.

Which of the following URLs share the same origin with "http://site.com"?

a) http://site.org

b) http://www.site.com

c) http://site.com/my/page.html

What is the role of the iframe element in web development?

a) It embeds a window within a separate document and window.

b) It defines a hyperlink within a web page.

c) It creates a new browser window.

What happens when a browser detects an iframe with a different origin than its parent window?

a) The browser automatically grants full access to the iframe.

b) Access to the iframe is denied by default.

c) The browser prompts the user for permission.

What does the sandbox attribute in an iframe element allow developers to do?

a) Exclude specific actions within the iframe to prevent execution from untrusted code.

b) Enable unrestricted access to the parent window.

c) Allow the iframe to access any resource on the internet.

Which of the following is a limitation imposed by the sandbox attribute?

a) allow-popups

b) allow-scripts

c) allow-top-navigation

In the context of nested iframes, what does window.frames refer to?

a) The collection of "children" windows.

b) The reference to the topmost parent window.

c) The reference to the "parent" (outer) window.

What is the purpose of the PostMessage interface in cross-window communication?

a) It allows windows to talk to each other regardless of origin, provided both sides accept the request.

b) It restricts communication between windows of different origins.

c) It encrypts messages sent between windows.

What information does the Event object include when handling messages sent via PostMessage?

a) Data, Origin, and Source

b) Sender, Receiver, and Payload

c) Timestamp, Location, and Type

How can one ensure that a message received via PostMessage is from the intended origin?

a) By checking the Origin property of the Event object.

b) By using a secure HTTPS connection.

c) By encrypting the message payload.

## Quiz 3

What is ClickJacking?

a) A technique that allows websites to steal user data.

b) A method that allows malicious pages to perform actions on a different site on behalf of the user.

c) A form of phishing attack.

What type of user interactions does ClickJacking primarily target?

a) Clicks and taps.

b) Keyboard inputs.

c) Voice commands.

How does ClickJacking occur?

a) By injecting malicious code into a victim's computer.

b) By luring a visitor to a malicious site and layering an invisible iframe over a catchy link.

c) By exploiting vulnerabilities in a website's server.

What is a common defense against ClickJacking that involves preventing a page from being opened within an iframe?

a) Framebusting.

b) Sandboxing.

c) Top-navigation blocking.

What does the X-Frame-Options header control?

a) How a webpage is displayed when embedded within an iframe on another site.

b) How a webpage handles user authentication.

c) How a webpage interacts with external APIs.

What is the value of X-Frame-Options that allows a page to be displayed in a frame only if the parent document is from the same origin?

a) SAMEORIGIN.

b) ALLOW-FROM.

c) DENY.

How can SameSite attribute in cookies help prevent ClickJacking attacks?

a) By ensuring that cookies are only sent when the user interacts directly with the site, not when embedded in an iframe on a different domain.

b) By encrypting the cookie data to make it more secure.

c) By blocking all cross-origin requests.

What value of SameSite attribute ensures that a cookie will only be sent with requests originating from the same site it was set on?

a) Strict.

b) Lax.

c) None.

Why is it important to be cautious when using SameSite=None for cookies?

a) Because it requires the cookie to be transmitted over HTTPS.

b) Because it allows the cookie to be sent with every request, regardless of the source.

c) Because it restricts the cookie to be used by the same site only.

Which of the following statements about ClickJacking is true?

a) ClickJacking primarily targets keyboard inputs.

b) ClickJacking is not visible to the user because iframes are invisible.

c) ClickJacking is a form of server-side attack.

## Quiz 4

1. What is an ArrayBuffer?

a) An array with dynamic length.

b) A fixed-length and fixed-memory-space object for storing raw binary data.

c) A type of array with specific data types.

2. How are individual bytes accessed in an ArrayBuffer?

a) Using views (TypedArray), not buffer[index].

b) Using buffer[index].

c) Using a special method called getByte(index).

3. Which of the following statements about TypedArray is true?

a) It is similar to arrays, but it cannot handle binary data.

b) It efficiently handles binary data and is useful for large numeric data tasks.

c) It is used exclusively for image processing.

4 What is the purpose of DataView?

a) To create a dynamic view of an ArrayBuffer.

b) To create a fixed-length array for storing binary data.

c) To allow access at any offset and format within an ArrayBuffer.

5 How does JavaScript handle out-of-range values in a TypedArray?

a) It throws an error.

b) It truncates the value, keeping the least significant bits.

c) Value is truncated, keeping least significant bits.

6 Which TypedArray is designed for image processing and automatically clamps values to the valid range (0-255)?

a) Uint8Array

b) Uint8ClampedArray

c) Uint32Array

7 What is the purpose of Uint8ClampedArray?

a) It is used for signed integers.

b) It is used for floating-point numbers.

c) It clamps values to the valid range (0-255) for image processing.

8 When creating a DataView, what does the parameter "byteOffset" represent?

a) The starting byte position (default 0).

b) The total number of bytes in the buffer.

c) The length of the DataView.

9 Which TypedArray treats every 2 bytes as an integer?

a) Uint16Array

b) Uint32Array

c) Uint64Array

10 What is the purpose of ArrayBuffer in the context of file operations and image processing?

a) It is used to efficiently handle binary data.

b) It is a fixed-length and fixed-memory-space object for storing raw binary data.

c) It is a dynamic array with variable length.

# File Objects

## Blob Object-

What is a Blob

A Blob is a data type representing a collection of binary data.

It's used for storing files, images, audio, video, and other binary data.

Blobs are immutable, raw data objects.

They're created using the Blob constructor or APIs that return Blob objects.

Blobs are useful when data doesn't need to be processed immediately but must be stored or transmitted in its raw form.

JavaScript provides the FileReader object to read Blob content.

Blobs can be converted into temporary URLs using URL.createObjectURL() for referencing in a browser.

**let content = ['Hello, World!'];**

**let blob = new Blob(content, { type: 'text/plain' });**

Blobs are essential for handling binary data in web applications, enabling efficient storage, manipulation, and transmission of files and media.

## File Object (upload file)-

Inherits from Blob and extends with filesystem capabilities.

Ways to obtain:

1. Constructor

**new File(fileParts, fileName, [options])**

* **fileParts:** Array of Blob/BufferSource/String values.
* **fileName:** Name of the file.
* **options (optional):**

lastModified: Timestamp (integer date) of last modification.

1. User interaction

**<input type="file">**

**drag-and-drop** or other browser interfaces.

Inherits file information from the operating system.

Getting a File object from **<input type="file">**

When **<input type="file">** is used, input.files is an array-like object containing selected files.

For a single file, it can be accessed as **input.files[0]**

The input.files property may contain multiple files if the input allows it. In this case, you can access each file using an **index**.

File object properties (in addition to Blob properties):

**name:** The file name.

**lastModified:** Timestamp of the last modification.

**<input type=”file” onchange=”showFile(this)”>**

**<script>**

**Function showFile(input) {**

**let file = input.files[0];**

**alert(`File name: ${file.name}`);**

**alert(`Last modified: ${file.lastModified}`);**

**}**

**</script>**

## File Reader

FileReader is designed for **reading data** from Blob (and File) objects. It **handles data delivery through events** due to potential time-consuming disk operations.

**let reader = new FileReader();**

Methods:

1. **readAsArrayBuffer(blob)**

Reads data in binary format as an ArrayBuffer.

For binary files and low-level binary operations. For high-level operations like slicing, File can be directly used, as it inherits from Blob.

1. **readAsText(blob, [encoding])**

Reads data as a text string with an optional encoding parameter (defaults to utf-8).

Suitable for text files when a string representation is desired.

1. **readAsDataURL(blob)**

Reads binary data and encodes it as a base64 data URL.

Used when embedding data in src attributes for tags like img. An alternative method for this operation is discussed in the Blob chapter: URL.createObjectURL(file)

1. **abort()**

Cancels the ongoing operation.

Events to occur as read proceeds

* loadstart: Indicates the start of loading.
* progress: Occurs during the reading process.
* load: No errors, reading is successfully completed.
* abort: Triggered when abort() is called.
* error: Signifies an error during the operation.
* loadend: Marks the end of reading, whether successful or not.
* reader.result: Contains the result (if successful).
* reader.error: Holds the error (if the operation fails).

Common events used: **Load** and **error**

**<input type=”file” onchange=”readFile(this)”>**

**<script>**

**function readFile(input) {**

**let file = input.files[0];**

**let reader = new FileReader();**

**reader.readAsText(file);**

**reader.onload = function() {**

**console.log(reader.result);**

**};**

**reader.onerror = function() {**

**console.log(reader.error);**

**};**

**}**

**</script>**

Additional Notes

FileReader is used to **read not only files but any blobs.**

* It converts a blob to various formats:
* readAsArrayBuffer(blob) converts to ArrayBuffer.
* readAsText(blob, [encoding]) converts to a string (an alternative to TextDecoder).
* readAsDataURL(blob) converts to a base64 data URL.

For Web Workers, there's a synchronous variant called FileReaderSync which returns results without generating events.

File objects inherit from Blob and include additional properties like name and lastModified.

FileReader objects can read from a file or a blob in three formats: String, ArrayBuffer, and Data URL (base64 encoded).

In many cases, there's no need to read file contents. Creating a short URL with URL.createObjectURL(file) allows easy referencing in HTML elements like <a> or <img>.

Sending a File over a network is straightforward as network APIs like XMLHttpRequest or fetch natively accept File objects.

## Fetch

JavaScript Network Requests

JavaScript allows sending network requests for tasks like submitting orders, fetching user info, or receiving updates without page reload.

The term "AJAX" (Asynchronous JavaScript And XML) refers to this, though XML isn't mandatory.

**fetch()** is a modern method for this task, offering versatility and solid browser support.

**let promise = fetch(url, [options])**

* **url: The target URL.**
* **options: Optional parameters**

How it works:

The browser initiates the request and returns a promise. Obtaining a response involves two stages. Firstly, a promise resolves with a Response object containing headers but not the body.

The promise rejects if there are network issues or the site doesn't exist.

**HTTP errors (e.g., 404) normal, won't cause rejection.**

Key response properties:

**ok:** true if HTTP status is 200-299.

**status:** HTTP status code.

## POST vs GET

A **POST** request is one of the HTTP methods used when a client (e.g., a browser) wants to send data to a server to create or update a resource on the server. This data is typically sent in the body of the request.

Unlike a **GET** request which appends data to the URL (and is primarily used for retrieving information), a POST request includes the data in the body of the request, which makes it suitable for sending larger amounts of data, like form submissions or file uploads.

For example, when you submit a form on a webpage, it's often sent as a POST request. This way, the data isn't visible in the URL, which is important for sensitive information like passwords.

## POST Requests

Making a POST Request with Fetch

To perform a POST request, or any other HTTP method, we employ fetch options:

**Method**: Specifies the HTTP method, e.g., POST.

**Body**: Can be:

* A string (e.g., JSON).
* FormData object, for submitting data as form/multipart.
* Blob/BufferSource for sending binary data.
* URLSearchParams, for submitting data in x-www-form-urlencoded encoding (rarely used).

**let user = { name: “John”, surname: “Smith” };**

**let response = await fetch(‘/article/fetch/post/user’, {**

**method: ‘POST’,**

**headers: {**

**‘Content-Type’: ‘application/json;charset=utf-8’**

**},**

**body: JSON.stringify(user)**

**});**

**let result = await response.json();**

**alert(result.message);**

When the body is a string, the default Content-Type is set to text/plain;charset=UTF-8. To send JSON-encoded data, we utilize the headers option to set it to application/json, which is the correct content type for JSON data.

This is the process of making a POST request using the fetch API and the associated options.

## Sending an Image

We can also submit binary data directly using Blob or BufferSource.

(code)

Response Properties

**response.status** – HTTP code of the response.

**response.ok** – True if the status is in the range of 200-299.

**response.headers** – A Map-like object containing HTTP headers.

Methods for Retrieving Response Body

**response.json()** – Parses the response as a JSON object.

**response.text()** – Returns the response as text.

**response.formData()** – Returns the response as a FormData object (for form/multipart encoding).

**response.blob()** – Returns the response as a Blob (binary data with type).

**response.arrayBuffer()** – Returns the response as an ArrayBuffer (pure binary data).

Fetch Options Recap

**method** – Specifies the HTTP method.

**headers** – An object with request headers (note that not all headers are allowed).

**body** – Can be a string, FormData, BufferSource, Blob, or UrlSearchParams object for sending data.

## GitHub GetUsers Program

Instructions

Create an async function getUsers(names), that gets an array of GitHub logins, fetche the users from GitHub and returns an array of GitHub users.

The GitHub url with user informaiton for the given USERNAME is: <https://api.github.com/users/USERNAME>.

Important details:

There should be one fetch request per user. And requests shouldn’t wait for each other. So that the data arrives as soon as possible.

If any request fails, or if there’s no such user, the function should return null in the resulting array.

**const usernames = ['octocat', 'nonexistentuser', 'anotheruser'];**

**const users = await getUsers(usernames);**

**console.log(users);**

**async function getUsers(names) {**

**const promises = names.map(async (name) => {**

**const response = await fetch(`https://api.github.com/users/${name}`);**

**if (response.ok) {**

**const user = await response.json();**

**return user;**

**} else {**

**return null;**

**}**

**});**

**const users = await Promise.all(promises);**

**return users;**

**}**

How it works

async function getUsers(names): This defines an asynchronous function named getUsers that takes an array of GitHub logins (names) as an argument.

const promises = names.map(async (name) => { ... }: Here, we're using map to create an array of promises. For each name, we perform an asynchronous operation.

const response = await fetch(https://api.github.com/users/${name}`);`: This line fetches the user information from GitHub for the provided username.

if (response.ok) { ... } else { ... }: Checks if the response is successful (status code 200-299). If it is, we proceed to parse the JSON response.

const user = await response.json();: This line extracts the JSON data from the response.

return user;: If everything is successful, we return the user information.

return null;: If there's an error or the user doesn't exist, we return null.

const users = await Promise.all(promises);: We wait for all the promises to resolve using Promise.all. This ensures that we get an array of users or null values.

return users;: Finally, we return the array of users.

# Form Data

## Simple Form

Sending HTML forms with or without files, additional fields etc.

Constructor

**let formData = new FormData([form]);**

FormData is an object to store and send form data. Network methods, such as fetch, can accept a FormDataobject as a body.

It’s encoded and sent out with Content-Type: form/multipart. So, from the server point of view, that looks like a usual form submission.

The server accepts the POST request with the form and replies “User saved”.

**<form id=”formElem”>**

**<input type=”text” name=”name” value=”John”>**

**<input type=”text” name=”surname” value=”Mike”>**

**<input type=”submit”>**

**</form>**

**<script>**

**formElem.onsubmit = async (e) => {**

**e.preventDefault();**

**let response = await fetch(‘/article/formdata/post/ user’, {**

**method: ‘POST’,**

**body: new FormData(formElem)**

**});**

**Let result = await response.json();**

**Alert(result.message);**

**};**

**</script>**

## FormData Methods

**formData.append(name, value):** Adds a form field with the given name and value.

**formData.append(name, blob, fileName):** Adds a field as if it were <input type="file">. The third argument fileName sets the file name.

**formData.delete(name):** Removes the field with the given name.

**formData.get(name):** Gets the value of the field with the given name.

**formData.has(name):** Checks if a field with the given name exists.

**formData.set(name, value) / formData.set(name, blob, fileName):** Removes all fields with the given name, then appends a new field.

Iterating over formData fields can be done using a for..of loop.

**let formData = new FormData();**

**formData.append(‘key1’, ‘value1’);**

**formData.append(‘key2’, ‘value2’);**

**for(let [name, value] of formData) {**

**alert(`${name} = ${value}`);**

**}**

## Sending a form with a file

The form is always sent as Content-Type: form/multipart.

This encoding allows to send files. So, <input type="file"> fields are sent also, similar to a usual form submission.

**<form id=”formElem”>**

**<input type=”text” name=”firstName” value=”John”>**

**Picture: <input type=”file” name=”picture” accept=”image/\*”>**

**<input type=”submit”>**

**</form>**

**<script>**

**formElem.onsubmit = async (e) => {**

**e.preventDefault();**

**let response = await fetch(‘/article/formdata/post/user-avatar’, {**

**method: ‘POST’,**

**body: new Formdata(formElem)**

**});**

**let result = await response.json();**

**alert(result.message);**

**};**

**</script>**

## \*Sending a form with Blob data

Sending Images with Form Data

Dynamically generated Blobs, like images, can be directly sent with a fetch request.

Convenience of Form Data

It's often more convenient to include an image as part of a form along with additional fields and metadata.

Server Compatibility

Servers are typically designed to handle multipart-encoded forms, making them a preferred choice for submitting images.

Submitting Images from Canvas

To submit an image from a <canvas> along with other fields, FormData can be employed.

Adding an Image Blob

Use **formData.append("image", imageBlob, "image.png")** to mimic the submission of a file named "image.png" as if it were from the visitor's filesystem.

(code)

**FormData** objects are used to capture HTML form and submit it using fetch or another network method.

We can either create new FormData(form) from an HTML form, or create an empty object, and then append fields with methods:

* formData.append(name, value)
* formData.append(name, blob, fileName)
* formData.set(name, value)
* formData.set(name, blob, fileName)

Two peculiarities here:

The set method removes fields with the same name, append doesn’t.

To send a file, 3-argument syntax is needed, the last argument is a file name, that normally is taken from user filesystem for **<input type="file">**

Other methods are:

* formData.delete(name)
* formData.get(name)
* formData.has(name)

## Fetch: Download Progress

Tracking Download Progress with fetch()

The fetch method provides a way to monitor download progress.

It currently does not support tracking upload progress.

Reading Response as a Stream

To track download progress, we utilize the response.body property. This represents a "readable stream" that provides the response body in chunks.

Managing Stream Chunks

We can control the reading process by handling the chunks of the response. Each chunk is an object with properties done (indicating if the reading is complete) and value (a typed array of bytes).

Reading Progress

**let response = await fetch(url);**

**let contentLength = +response.headers.get('Content-Length');**

**let receivedLength = 0;**

**let reader = response.body.getReader();**

**let chunks = [];**

**while (true) {**

**const { done, value } = await reader.read();**

**if (done) break;**

**chunks.push(value);**

**receivedLength += value.length;**

**console.log(`Received ${receivedLength} of ${contentLength}`);**

**}**

**let chunksAll = new Uint8Array(receivedLength);**

**let position = 0;**

**for (let chunk of chunks) {**

**chunksAll.set(chunk, position);**

**position += chunk.length;**

**}**

**let result = new TextDecoder().decode(chunksAll);**

**let data = JSON.parse(result);**

Handling Binary Content

If binary content is needed instead of JSON, it's simpler. Replace steps 4 and 5 with the following:

**let blob = new Blob(chunks);**

This will give you the binary content as a Blob.

This process is for tracking download progress, not upload progress. Upload progress tracking is not currently supported with fetch. For upload progress, consider using XMLHttpRequest.

## Fetch: Abort

Fetch returns a promise. And JavaScript generally has no concept of “aborting” a promise. So how can we cancel a fetch?

Aborting a fetch request can be achieved using the AbortController object. This is especially useful when we want to cancel an ongoing request.

Steps to Implement Aborting a Fetch Request:

1. Create an AbortController:

**let controller = new AbortController();**

The AbortController has two main properties: **abort()** method and **signal**.

2. Pass the signal Property to Fetch:

**fetch(url, {**

**signal: controller.signal**

**});**

This associates the controller's signal with the fetch request.

3. To Abort, Call controller.abort():

**controller.abort();**

This will trigger the abort event on the AbortController's signal.

Handelling Abort Error:

**try {**

**let response = await fetch(url, { signal: controller.signal });**

**let data = await response.json();**

**// Process the data**

**} catch (err) {**

**if (err.name === 'AbortError') {**

**console.log('Fetch aborted');**

**} else {**

**console.error('Error:', err);**

**}**

**}**

Aborting Multiple Fetches

**let controller = new AbortController();**

**let urls = ['url1', 'url2', 'url3'];**

**for (let url of urls) {**

**fetch(url, { signal: controller.signal })**

**.then(response => response.json())**

**.then(data => console.log(data))**

**.catch(err => {**

**if (err.name === 'AbortError') {**

**console.log('Fetch aborted');**

**} else {**

**console.error('Error:', err);**

**}**

**});**

**}**

**// To abort all fetches**

**controller.abort();**

# Fetch: Cross Origin Requests

## Cross-Origin Requests and CORS

Origin Concept

An origin in web development refers to a combination of domain, port, and protocol

**https://example.com:3000**

Cross-Origin Requests

These are requests sent from one domain to another. They include requests to different domains, subdomains, or those using different protocols or ports.

CORS (Cross-Origin Resource Sharing):

It's a security policy implemented by web browsers that controls how resources from one domain can interact with resources from a different domain.

**try {**

**await fetch('http://example.com');**

**} catch(err) {**

**alert(err);** // Failed to fetch due to restrictions

**}**

**Importance of CORS:** CORS prevents malicious scripts on one site from tampering with or stealing data from another site. It's a fundamental security measure on the internet.

**Browser Enforcement:** Modern web browsers enforce CORS policies, allowing developers to control which domains can access their resources.

**Server-Side Configuration:** Additionally, servers can be configured to include appropriate CORS headers, specifying which domains are permitted to access their resources.

## Using Forms

In the early days of web development, it was feasible to make GET/POST requests to other sites, even without dedicated networking methods.

However, accessing the content of an <iframe> from a different site was restricted, making it impossible to read the response.

Although forms could send data to any destination, they couldn't receive the response. Some workarounds existed, but they're considered outdated and not commonly used today.

// form target

**<iframe name=”iframe”></iframe>**

// form could be dynamically generated/submitted

**<form target=”iframe” method=”POST” action=”http://another.com/...”>**

**</form>**

## Simple Requests

Types of Cross-Domain Requests:

* Simple Requests.
* All others (considered "non-simple")

Conditions for a Simple Request:

* Simple methods allowed: **GET, POST, HEAD.**
* Only specific custom headers are allowed.

Custom headers allowed:

Accept,

Accept-Language,

Content-Language,

Content-Type with the value application/x-www-form-urlencoded, multipart/form-data or text/plain.

Simple Request vs Non-Simple Request:

**Simple requests** can be made with a <form> or a <script>, without special methods.

**Non-simple requests** involve non-standard headers or methods like DELETE and require special handling.

Preflight Request:

Sent by the browser when making a non-simple request.

Asks the server if it agrees to accept such cross-origin requests.

Purpose of Restrictions:

Ensure new cross-origin capabilities are only accessible with explicit permission from the server.

## CORS for simple requests

Making simple requests

When a cross-origin request is made, the browser adds an **Origin header** to it.

This header contains the origin **(domain/protocol/port)** without a path.

The server checks this Origin and can choose to accept it.

**If accepted**, it responds with an Access-Control-Allow-Origin header, specifying the allowed origin. If not, it results in an error.

The browser ensures the correct Origin is sent and checks for the appropriate Access-Control-Allow-Origin in the response. If everything aligns, JavaScript gets access, otherwise, it's blocked with an error.

For instance, if we request

https://anywhere.com/request from https://javascript.info/page, the headers will be like:

**GET /request**

**Host: anywhere.com**

**Origin: https://javascript.info**

This Origin contains exactly the origin (domain/protocol/port), without a path.

A diagram of a computer program

Description automatically generated

Here’s an example of a permissive server response:

**200 OK**

**Content-Type:text/html; charset=UTF-8**

**Access-Control-Allow-Origin: https://javascript.info**

## Response Headers

By default, JavaScript from one domain can only access certain response headers (known as "simple response headers").

These include headers like

* Cache-Control
* Content-Language
* Content-Type
* Expires
* Last-Modified
* Pragma

Notably, the **Content-Length header** is not included in this list.

If you want JavaScript to be able to access other response headers, the server must explicitly allow them using the Access-Control-Expose-Headers header.

**200 OK**

**Content-Type:text/html; charset=UTF-8**

**Content-Length: 12345**

**API-Key: 2c9de507f2c54aa1**

**Access-Control-Allow-Origin: https://javascript.info**

**Access-Control-Expose-Headers: Content-Length,API-Key**

With this Access-Control-Expose-Headers header, the script is **permitted to access the Content-Length** and API-Key headers of the response.

## Non-Simple Requests

Non-simple requests

Allow any HTTP method, like PATCH, DELETE, etc.

Older web services might see non-standard methods as a sign that it's not a browser.

To avoid misunderstandings, browsers send a "preflight" request before making a non-simple request.

The preflight request

uses the **OPTIONS** method and doesn't contain a body.

It includes headers like **Access-Control-Request-Method (for the requested method)** **and Access-Control-Request-Headers (for non-simple headers).**

If the server approves, it should respond with status 200, and include the headers Access-Control-Allow-Methods (with the allowed method) and Access-Control-Allow-Headers (with the allowed headers).

Optionally, Access-Control-Max-Age header can specify a number of seconds to cache the permissions, saving preflight requests for subsequent requests with the same permissions

A diagram of a computer program

Description automatically generated

**Let response = await fetch(‘https://site.com/services.json’, {**

**method: ‘PATCH’,**

**headers: {**

**‘Content-Type’: ‘application/json’**

**‘API-Key’: ‘secret’**

**}**

**});**

**Understanding Cross-Domain PATCH Request:**

Step 1 (Preflight Request):

Browser sends an invisible preflight request automatically.

This request has method OPTIONS with the same path as the main request.

It includes special headers like Origin, Access-Control-Request-Method, and Access-Control-Request-Headers.

Step 2 (Preflight Response):

Server responds with status 200 and headers specifying allowed methods and headers.

In this example, it's PATCH method and headers Content-Type, API-Key.

Note: If the server expects other methods or headers in the future, it can allow them by adding to the list.

Step 3 (Actual Request):

Browser makes the real PATCH request after a successful preflight.

It includes headers like Origin, Content-Type, API-Key.

Step 4 (Actual Response):

Server includes Access-Control-Allow-Origin in the response header.

This allows JavaScript to read the full response.

Important Notes:

Preflight request is automatic and invisible to JavaScript.

JavaScript only gets the response to the main request or an error if there's no server permission.

## Credentials

Cross-Origin Requests and Credentials:

Cross-origin requests **do not** include credentials (like cookies) by default.

This is a security measure to prevent unauthorized access.

To allow credentials, the server must explicitly permit it with an additional header.

Sending Credentials:

**fetch('http://another.com', {**

**credentials: "include"**

**});**

**Server Response for Credentials:** Access-Control-Allow-Credentials: true

Simple Requests vs. Non-Simple Requests:

Simple requests include methods like GET, POST, or HEAD, and specific headers.

Non-simple requests include other methods and headers, requiring a preliminary "preflight" request.

Handling Simple Requests:

The browser sends the Origin header the request.

The server should set Access-Control-Allow-Origin to the origin.

**Access-Control-Allow-Origin: <origin>**

**Access-Control-Allow-Credentials: true**

Accessing Non-Simple Response Headers:

If JavaScript needs to access specific response headers, the server should list them in Access-Control-Expose-Headers.

Why Do We Need Origin?

Origin is more reliable than Referer, especially in cross-origin requests.

Referer can be absent in some cases, but Origin is guaranteed by the browser.

Solutions:

Origin is used as a reliable source of the request's origin.

It is necessary for security, especially in cases where Referer might not be sent.

## Fetch API

List of all fetch options

A screenshot of a computer program

Description automatically generated

**Referrer**: Controls the HTTP Referer header.

* To send no referer, use an empty string or "no-referrer".
* To set another URL within the current origin, provide the URL.

**fetch('https://example.com/page', {**

**referrer: ''**

**});**

**fetch('https://example.com/page', {**

**referrer: 'https://example.com/otherpage'**

**});**

**ReferrerPolicy:** Sets general rules for Referer.

* Options include "no-referrer-when-downgrade", "no-referrer", "origin", and others.

Mode:

* Determines the behavior for cross-origin requests:
  + **"cors"**: Allows cross-origin requests (default).
  + **"same-origin"**: Forbids cross-origin requests.
  + **"no-cors"**: Only allows simple cross-origin requests.

Credentials:

* Specifies whether fetch should send cookies and HTTP-Authorization headers with the request.
  + **"same-origin"**: Default, don’t send for cross-origin requests.
  + **"include"**: Always send, requires **Access-Control-Allow-Credentials** from the cross-origin server.
  + **"omit"**: Never send, even for same-origin requests.

Cache:

* Fine-tunes HTTP caching behavior.
  + **"default"**: Uses standard HTTP-cache rules and headers.
  + **"no-store"**: Totally ignores HTTP-cache.
  + **"reload"**: Ignores HTTP-cache but populates it with the response.
  + **"no-cache"**: Creates a conditional request if there is a cached response.
  + **"force-cache"**: Uses a response from HTTP-cache, even if it’s stale.
  + **"only-if-cached"**: Uses a response from HTTP-cache, even if it’s stale.

Redirect:

* Controls how fetch handles HTTP-redirects:
  + **"follow"**: Follow HTTP-redirects (default).
  + **"error"**: Throws an error in case of HTTP-redirect.
  + **"manual"**: Doesn’t follow HTTP-redirect, but provides information for manual redirection.

Integrity:

* Checks if the response matches a known-ahead checksum.
  + Supports hash-functions like SHA-256, SHA-384, and SHA-512.

keepalive:

* Indicates that the request may outlive the page.
  + Essential for requests to succeed even after the page is unloaded.

**window.onunload = function() {**

**fetch('/analytics', {**

**method: 'POST',**

**body: "statistics",**

**keepalive: true**

**});**

**};**

Note: Each option serves specific purposes, allowing fine-grained control over the behavior of the fetch request.

# Patterns and Flags

## Syntax

Long Syntax: **regexp = new RegExp("pattern", "flags");**

Short Syntax with slashes: **regexp = /pattern/;**

## Searching with search

**let str = "I love JavaScript!";**

**let regexp = /love/;**

**alert( str.search(regexp) );** //2

**/love/** is a simple substring search

## Using new RegExp

Allows for dynamic pattern construction

**let tag = prompt("Which tag you want to search?", "h2");**

**let regexp = new RegExp(tag);**

**alert( "<h1> <h2> <h3>".search(regexp));** // Finds **<h2>** by default

## Flags

* **i**: Case-insensitive search
* **g**: Looks for all matches
* **m**: Multiline mode
* **s**: Dotall mode
* **u**: Enables full unicode support
* **y**: Sticky mode

## Methods

1. Searching for all matches

Using g flag:

Get flat array of matches: str.match(reg)

Get array of matches with details: str.matchAll(reg)

2. Searching for the first match only:

Without g flag:

Get the full first match: str.match(reg)

Get the position of the first match: str.search(reg)

Check if there's a match: regexp.test(str)

Find the match from a given position: regexp.exec(str)

3. Replacing all matches:

Replace with another string or a function result: str.replace(reg, str|func)

4. Splitting the string by a separator:

str.split(str|reg)

## Misc properties

Word Boundaries

\d: Any digit (0-9).

\s: Any whitespace character.

\w: Any word character (letter, digit, underscore).

\D: Any non-digit.

\S: Any non-whitespace character.

\W: Any non-word character.

Dot .

Matches any character except a newline.

Flags

s: Makes the dot match newlines.

Escaping Special Characters

To match a special character literally, escape it with a backslash (\).

## Examples

**let str = "I love JavaScript!";**

**alert( str.search(/LOVE/i) );** // 2 (found lowercased)

**alert( str.search(/LOVE/) );** // -1 (nothing found without 'i' flag)

**let str = "I love JavaScript!";**

**let matchResult = str.match(/love/);**

// Result: ["love", index: 2, input: "I love JavaScript!"]

**let str = "I love JavaScript!";**

**let matchResult = str.match(/love/g);**

// Result: ["love"]

**let str = "Hello, John Doe!";**

**let newStr = str.replace(/John/g, 'Mr.$&');**

// Result: "Hello, Mr.John Doe!"

**let regexp = /love/gi;**

**let str = "I love JavaScript";**

**regexp.lastIndex = 10;**

**let match = regexp.exec(str);**

// Result: ["love"]

**let regexp = /love/gi;**

**let str = "I love JavaScript";**

**regexp.lastIndex = 10;**

**alert( regexp.test(str) );** // false (no match)

# Quizzes

## File Objects, Blobs, and Fetch

**1. What is a Blob in JavaScript?**

A. A data type for storing text data  
B. A data type representing a collection of binary data  
C. A method for fetching external resources

**2. When would you use a Blob in web development?**

A. For storing only text-based data  
B. For handling binary data like files, images, audio, etc.  
C. For creating animations on a webpage

**3. How can you convert a Blob into a temporary URL for use in a browser?**

A. Using URL.createObjectURL()  
B. Using FileReader.readAsDataURL()  
C. Using Blob.toURL()

**4. How do you create a Blob in JavaScript?**

A. **let blob = new Blob(data, type);**  
B. **let blob = Blob.create(data, type);**  
C. **let blob = createBlob(data, type);**

**5. What is a File object and how does it relate to Blobs?**

A. A File object is used to handle text data, while Blobs handle binary data.  
B. A File object is a specialized Blob with added filesystem capabilities.  
C. A File object is a type of Blob that is only used for images.

**6. How can you obtain a File object in JavaScript?**

A. By calling **new File(data, name, options)**  
B. By using an **<input type="file">** element or other browser interfaces.  
C. By using the **File.create(data, name)** method.

**7. What properties are unique to a File object compared to a Blob?**

A. **size** and **type**  
B. **name** and **lastModified**  
C. **filename** and **extension**

**8. What is the purpose of the FileReader object in JavaScript?**

A. It is used for reading and writing text files.  
B. It is designed for reading data from Blob and File objects.  
C. It is used for creating and manipulating Blobs.

**9. Which method of FileReader would you use to read data as binary?**

A. **readAsArrayBuffer(blob)**  
B. **readAsText(blob, encoding)**  
C. **readAsDataURL(blob)**

**10. What is the purpose of the fetch() method in JavaScript?**

A. To retrieve network requests and handle them asynchronously.  
B. To create new files on a server.  
C. To draw graphics on a webpage.

**11. Which Blob method allows you to slice a portion of the Blob into a new Blob?**

A. **slice(start, end, contentType)**  
B. **cut(start, end)**  
C. **portion(start, end)**

**12. What is the purpose of the load event in FileReader?**

A. It occurs when the reading process starts.  
B. It occurs during the reading process, providing progress information.  
C. It signifies that the reading process is successfully completed.

**13. What does the abort() method of FileReader do?**

A. It cancels the ongoing file upload process.  
B. It cancels the ongoing read operation.  
C. It aborts the current JavaScript program execution.

**14. When might you use the readAsDataURL(blob) method of FileReader?**

A. When you want to read data as binary.  
B. When you want to encode binary data as a base64 data URL.  
C. When you want to read data as text.

**15. What property of a File object provides the last modification timestamp?**

A. **modificationTime**  
B. **lastModified**  
C. **timestamp**

**16. What is the purpose of the FormData object in JavaScript?**

A. It is used for creating forms in HTML.  
B. It is used for sending form data in an HTTP request.  
C. It is used for formatting date and time.

**17. Which HTTP method is typically used for sending sensitive data like passwords?**

A. GET  
B. POST  
C. PUT

**18. When making a POST request, where is the data usually placed?**

A. In the URL  
B. In the request body  
C. In the request headers

**19. Which response property contains the HTTP status code?**

A. **statusCode**  
B. **status**  
C. **httpCode**

**20. What does the response.ok property indicate?**

A. Whether the request was successful (status code 200-299)  
B. Whether the request was made with the correct method  
C. Whether the response was empty

**21. Which method of the Response object is used for parsing the response as JSON?**

A. **parseJSON()**  
B. **toJSON()**  
C. **json()**

**22. How can you send binary data using fetch?**

A. By converting it to a base64 data URL  
B. By using the FormData object  
C. By using Blob or BufferSource

**23. What is the purpose of the response.text() method?**

A. It returns the response as text.  
B. It converts the response to uppercase.  
C. It extracts the response as JSON.

**24. How do you cancel an ongoing fetch request?**

A. By calling **request.abort()**  
B. By using the **cancel()** method  
C. By sending a DELETE request

**25. In the context of fetch, what does a HTTP status code in the 300-399 range usually indicate?**

A. A successful response  
B. A redirection  
C. An error

**26. What is the purpose of the FormData.append() method?**

A. It appends data to the URL of a FormData object.  
B. It appends a new value onto an existing key inside a FormData object.  
C. It appends a new FormData object to an existing one.

**27. How do you send form data using fetch()?**

A. By serializing the form data to a JSON object  
B. By creating a FormData object and passing it as the request body  
C. By manually concatenating the form fields as a string

**28. Which method of Response retrieves the response as a Blob?**

A. **response.blob()**  
B. **response.text()**  
C. **response.json()**

**29. What is the primary difference between a Blob and a File object?**

## FormData

**1. What is the purpose of FormData in JavaScript?**

A. To encode form data for submission  
B. To create dynamic forms  
C. To validate form inputs

**2. Which method is used to add a form field to a FormData object?**

A. formData.add(name, value)  
B. formData.append(name, value)  
C. formData.insert(name, value)

**3. How can you remove a field from a FormData object?**

A. formData.remove(name)  
B. formData.delete(name)  
C. formData.erase(name)

**4. What is the purpose of the get method in a FormData object?**

A. It retrieves the value of a field with the given name.  
B. It checks if a field with a given name exists.  
C. It gets the total number of fields in the FormData object.

**5. When is the 3-argument syntax used for the append method in FormData?**

A. When adding a field with a blob (like a file)  
B. When adding a text field  
C. When adding a numeric field

**6. How can you check if a field with a specific name exists in a FormData object?**

A. formData.exists(name)  
B. formData.includes(name)  
C. formData.has(name)

**7. What is the primary purpose of using FormData with the fetch API?**

A. To send HTML forms with or without files  
B. To fetch external resources like images and scripts  
C. To create dynamic forms

**8. What is the Content-Type used when FormData is sent in a fetch request?**

A. application/json  
B. application/x-www-form-urlencoded  
C. multipart/form-data

**9. How do you track download progress with the fetch API?**

A. By using the **response.progress** property  
B. By reading the response body as a stream  
C. By using the **download** event

**10. Which method is used to abort a fetch request?**

A. **abort()**  
B. **cancel()**  
C. **stop()**

**11. What is the purpose of the AbortController object in fetch requests?**

A. To track download progress  
B. To handle stream chunks  
C. To abort an ongoing fetch request

**12. How can you handle an aborted fetch request in a try-catch block?**

A. By checking if **err.type === 'AbortError'**  
B. By using **err.abort === true**  
C. By using **err.name === 'AbortError'**

**13. When might you use FormData in combination with a canvas element?**

A. To submit an image along with other form fields  
B. To create dynamic forms  
C. To draw graphics on a webpage

**14. What is the primary benefit of using FormData for image submission?**

A. It allows the inclusion of additional fields and metadata along with the image.  
B. It ensures that the image is always submitted as a file.  
C. It improves server compatibility.

**15. How do you add an image Blob to FormData for submission?**

A. **formData.append("image", imageBlob)**  
B. **formData.addImage("image", imageBlob)**  
C. **formData.setImage("image", imageBlob)**

**16. In the context of FormData, what does the set method do?**

A. It removes all fields with the given name and appends a new field.  
B. It appends a new value onto an existing key inside a FormData object.  
C. It appends a new FormData object to an existing one.

**17. Which Content-Type is used when FormData is sent in a fetch request without files?**

A. application/json  
B. application/x-www-form-urlencoded  
C. text/plain

**18. How can you send dynamically generated Blobs with a fetch request?**

A. By appending them to a FormData object  
B. By converting them to base64 data URLs  
C. By using the FileReader object

**19. What is the main advantage of using FormData for file uploads?**

A. It simplifies the encoding process of files.  
B. It allows additional fields to be included in the submission.  
C. It ensures that files are always submitted as Blobs.

**20. When using FormData, how do you send files along with additional form fields?**

A. By appending both the file and form fields to the FormData object  
B. By using separate fetch requests for each field  
C. By sending the form fields in the request headers

**21. Which property of FormData is used to track the progress of downloading a response?**

A. **body**  
B. **progress**  
C. **downloadProgress**

**22. What does the response.body.getReader() method return in a fetch response?**

A. A readable stream  
B. The response body as a string  
C. An array of chunks

**23. What is the purpose of using a while loop when reading a response body as a stream?** A. To concatenate all chunks into a single string  
B. To handle each chunk of the response body  
C. To convert the response body to JSON

**Answer: B**

**24. How do you track the total progress of a response download?** A. By using the **response.total** property  
B. By keeping count of the total chunks received  
C. By using the **Content-Length** header

**Answer: C**

**25. When might you use the chunksAll.set(chunk, position) method?** A. To concatenate chunks into a single string  
B. To append chunks to a Blob  
C. To track the total progress of a response download

**Answer: B**

**26. What is the purpose of using the TextDecoder in response processing?** A. To convert binary data to a readable string  
B. To parse JSON responses  
C. To handle stream chunks

**Answer: A**

**27. When is it more suitable to use XMLHttpRequest for upload progress tracking instead of fetch?** A. When using FormData for image submission  
B. When sending large files with FormData  
C. When sending text data with FormData

**Answer: B**

**28. How can you cancel multiple fetch requests at once?** A. By calling **controller.stopAll()**  
B. By creating a new AbortController for each request  
C. By calling **controller.abort()** once

**Answer: B**

**29. What happens when you call controller.abort() in a fetch request?** A. It triggers the abort event on the AbortController's signal.  
B. It cancels the ongoing request and rejects the associated promise

## Fetch: Cross Origin Requests and CORS

**1. What is an "origin" in web development?**

A. A combination of domain, port, and protocol  
B. A specific subdomain  
C. The content of an iframe

**2. What does CORS stand for?**

A. Cross-Origin Request Standard  
B. Cross-Origin Resource Sharing  
C. Cross-Origin Response System

**3. Why is CORS important in web development?**

A. It allows any script to access resources from any domain.  
B. It prevents malicious scripts from tampering with or stealing data from other sites.  
C. It enforces a specific protocol for cross-origin requests.

**4. How is CORS enforced in web browsers?**

A. By adding an "Origin" header to cross-origin requests  
B. By using a special fetch option called "credentials"  
C. By configuring the server to allow specific domains

**5. Which server-side configuration is used to specify which domains can access its resources?**

A. Access-Control-Allow-Headers  
B. Access-Control-Allow-Origin  
C. Access-Control-Allow-Methods

**6. In early web development, how were forms used to interact with other sites?**

A. They could send data but couldn't receive the response.  
B. They could receive the response but couldn't send data.  
C. They had no limitations in interacting with other sites.

**7. What is a "simple request" in the context of cross-origin requests?**

A. A request made using only the GET method.  
B. A request made using any method other than GET, POST, or HEAD.  
C. A request that includes specific custom headers.

**Answer: A**

**8. Which headers are allowed in a simple request?** A. Accept, Accept-Language  
B. Content-Type (with specific values), Content-Language  
C. Authorization, Content-Type

**Answer: B**

**9. When is a preflight request sent by the browser?** A. Before making a non-simple request  
B. Before making a simple request  
C. Only when custom headers are included

**Answer: A**

**10. What is the purpose of the preflight request in CORS?** A. To ask the server if it allows the cross-origin request  
B. To fetch additional information about the requested resource  
C. To check if the requested method is supported by the server

**Answer: A**

**11. What does the Access-Control-Expose-Headers header do?** A. It lists the headers that JavaScript is allowed to access.  
B. It specifies which headers should be included in the response.  
C. It controls the caching of permissions for subsequent requests.

**Answer: A**

**12. When making a non-simple request, which HTTP method can be used?** A. PATCH  
B. GET  
C. POST

**Answer: A**

**13. In a preflight request, which HTTP method is used?** A. OPTIONS  
B. HEAD  
C. TRACE

**Answer: A**

**14. What is the purpose of the credentials option in a fetch request?** A. To specify the allowed credentials for the request  
B. To indicate whether cookies and authorization headers should be sent  
C. To set the level of access to response headers

**Answer: B**

**15. When would you use credentials: "include" in a fetch request?** A. When you want to allow any HTTP method  
B. When you want to send cookies and authorization headers with the request  
C. When you want to forbid cross-origin requests

**Answer: B**

**16. What is the default behavior of a fetch request with regard to cookies and authorization headers?** A. They are always sent with the request.  
B. They are never sent with the request.  
C. They are sent only for same-origin requests.

**Answer: C**

**17. Which fetch option is used to fine-tune HTTP caching behavior?** A. Cache  
B. Redirect  
C. ReferrerPolicy

**Answer: A**

**18. What does the ReferrerPolicy option control?** A. The behavior of HTTP redirects  
B. The rules for the HTTP Referer header  
C. The caching behavior of the response

**Answer: B**

**19. When might you use mode: "no-cors" in a fetch request?** A. When you want to prevent any cross-origin requests  
B. When you want to allow only simple cross-origin requests  
C. When you want to use standard HTTP-cache rules

**Answer: A**

**20. How does the browser enforce CORS policies?** A. By adding an "Origin" header to cross-origin requests and checking the response headers  
B. By sending a preflight request for every cross-origin request  
C. By automatically allowing all cross-origin requests

**Answer: A**

**21. What does the "keepalive" option in a fetch request indicate?** A. That the request may outlive the page  
B. That the request will be terminated before the page is unloaded  
C. That the request will be delayed until the page is fully loaded

**Answer: A**

**22. How can you send a fetch request that includes cookies and authorization headers?** A. By setting **credentials: "include"**  
B. By adding them directly to the request headers  
C. By using the **withCredentials** property in the request

**Answer: A**

**23. Why is the "Origin" header considered more reliable than the "Referer" header for cross-origin requests?** A. "Referer" can be absent in some cases, while "Origin" is always sent.  
B. "Origin" contains

## Regular Expressions Quiz

**Question 1: What is a regular expression?**

A. A programming language  
B. A pattern matching tool  
C. A data structure  
D. A mathematical equation

**Question 2: Which syntax is used to create a regular expression object?**

A. Long Syntax: **regex = new RegExp("pattern", "flags")**  
B. Short Syntax: **regex = /pattern/**  
C. Both A and B  
D. Neither A nor B

**Question 3: What does the method str.search(reg) do?**

A. Searches for all matches in a string  
B. Searches for the first match only  
C. Replaces all matches  
D. Splits the string by a separator

**Question 4: When would you use new RegExp instead of the short syntax /.../?**

A. When you need to perform a case-insensitive search  
B. When you want to construct a pattern dynamically from a string  
C. When you want to find all matches in a string  
D. When you want to split a string by a separator

**Question 5: Which flag makes a search case-insensitive?**

A. **i**  
B. **g**  
C. **m**  
D. **s**

**Question 6: What does the flag g do in a regular expression?**

A. Searches for all matches, not just the first one  
B. Performs a case-insensitive search  
C. Enables full unicode support  
D. Allows **.** to match newlines

**Question 7: Which method is used to search for all matches with all details?**

A. **str.match(reg)**  
B. **str.matchAll(reg)**  
C. **str.search(reg)**  
D. **regexp.exec(str)**

**Question 8: What does the dot . match in a regular expression?**

A. Any character except a newline  
B. Any digit  
C. Any whitespace character  
D. Any non-word character

**Question 9: What is the purpose of the word boundary \b in a regular expression?**

A. Matches a boundary between characters  
B. Matches any whitespace character  
C. Matches a digit  
D. Matches a standalone word

**Question 10: When do you need to escape a special character in a regular expression?**

A. To match it literally  
B. To perform a case-insensitive search  
C. To enable full unicode support  
D. To find all matches

**Question 11: What does the flag s do in a regular expression?**

A. Enables full unicode support  
B. Allows **.** to match newlines  
C. Searches for all matches  
D. Performs a case-insensitive search

**Question 12: Which character class matches any word character (letter, digit, underscore)?**

A. **\w**  
B. **\d**  
C. **\s**  
D. **\W**

**Question 13: How do you search for a literal dot in a regular expression?**

A. **.**  
B. **\.**  
C. **\d**  
D. **\w**

**Question 14: Which flag is used to make the dot match newlines?**

A. **s**  
B. **i**  
C. **g**  
D. **m**

**Answer: A**. **s**.

**Question 15: When using new RegExp, why do you need to double backslashes (\\)?**

A. To match a special character literally  
B. To enable full unicode support  
C. Because strings consume one backslash  
D. To perform a case-insensitive search

**Answer: C**. Because strings consume one backslash.

**Question 16: Which method is used to replace matches in a string?**

A. **str.match(reg)**  
B. **str.replace(reg, str|func)**  
C. **str.search(reg)**  
D. **regexp.test(str)**

**Answer: B**. **str.replace(reg, str|func)**.

**Question 17: Which flag allows a search to start from a given position?**

A. **g**  
B. **i**  
C. **y**  
D. **m**

**Answer: C**. **y**.

**Question 18: What does the flag m do in a regular expression?**

A. Enables full unicode support  
B. Searches for all matches  
C. Multiline mode  
D. Allows **.** to match newlines

**Answer: C**. Multiline mode.

**Question 19: What does the method str.match(reg), no "g" flag return?**

A. An array of all matches  
B. The position of the first match  
C. The first match with details  
D. An array of the first match

**Answer: C**. The first match with details.

**Question 20: What is the purpose of character class \D in a regular expression?**

A. Matches any whitespace character  
B. Matches any non-digit  
C. Matches any word character  
D. Matches any non-word character

**Answer: B**. Matches any non-digit.

**Question 21: Which flag allows the dot to match newlines?**

A. **s**  
B. **g**  
C. **m**  
D. **i**

**Answer: A**. **s**.

**Question 22: When do you need to escape a slash / in a regular expression?**

A. When using **/.../** to create a regexp  
B. When using **new RegExp** to create a regexp  
C. When searching for a word boundary  
D. When searching for a whitespace character

**Answer: A**. When using **/.../** to create a regexp.

**Question 23: Which method is used to find the position of the first match?**

A. **str.match(reg)**  
B. **str.search(reg)**  
C. **regexp.exec(str)**  
D. **regexp.test(str)**

**Answer: B**. **str.search(reg)**.

**Question 24: What does the flag u do in a regular expression?**

A. Enables full unicode support  
B. Searches for all matches  
C. Sticky mode  
D. Multiline mode

**Answer: A**. Enables full unicode support.

**Question 25: Which method is used to check if there's a match in a string?**

A. **str.match(reg)**  
B. **str.search(reg)**  
C. **regexp.exec(str)**  
D. **regexp.test(str)**

**Answer: D**. **regexp.test(str)**.

**Question 26: Which flag enables full unicode support in a regular expression?**

A. **u**  
B. **g**  
C. **i**  
D. **s**

**Answer: A**. **u**.

**Question 27: What does the flag y do in a regular expression?**

A. Sticky mode  
B. Multiline mode  
C. Enables full unicode support  
D. Allows **.** to match newlines

**Answer: A**. Sticky mode.

**Question 28: Which method is used to get an array of all matches with details?**

A. **str.match(reg)**  
B. **str.matchAll(reg)**  
C. **str.search(reg)**  
D. **regexp.exec(str)**

**Answer: B**. **str.matchAll(reg)**.

**Question 29: Which method is used to get an array of all matches without details?**

A. **str.match(reg)**  
B. **str.matchAll(reg)**  
C. **str.search(reg)**  
D. **regexp.exec(str)**

**Answer: A**. **str.match(reg)**.

**Question 30: Which flag is used to find all matches in a string?**

A. **g**  
B. **i**  
C. **y**  
D. **m**

**Answer: A**. **g**.

# **NodeJS Introduction**

## What is NodeJS

JavaScript is primarily used in web pages and gaming applications, often in conjunction with other programming languages.

Node JS is a runtime environment for executing JavaScript code outside of a web browser. It allows for server-side programming. Node JS is popular in the industry for its ability to create cross-platform applications compatible with Android and iOS devices.

## More on NodeJS

Node.js is an asynchronous, event-driven JavaScript runtime designed for building scalable network applications. It **handles multiple connections** concurrently and **sleeps when there's no work to be done**, contrasting with thread-based networking which is less efficient. Node.js users don't have to worry about process deadlocks since there are no locks.

Node.js is influenced by systems like Ruby's Event Machine or Python's Twisted, but it takes the event model further by presenting an event loop as a runtime construct. It doesn't require a blocking call to start the event loop. HTTP is a primary focus in Node.js, designed for streaming and low latency.

Although Node.js doesn't use threads, it can still utilize multiple cores. Child processes can be spawned, and the cluster module allows for socket sharing to enable load balancing over cores.

Node.js is built against modern versions of the V8 engine, ensuring timely delivery of new ECMAScript features. These features are split into three groups: shipping (default), staged (require a runtime flag), and in progress (discouraged for production use).

To find out which features are available in each Node.js release, you can use the --v8-options argument and grep for in-progress features, but be cautious as they may be incomplete or broken. The V8 team works on optimizing new language features to reach performance parity with their ES5 counterparts, tracked on the six-speed website.

The V8 team also coordinates improvement efforts through a performance plan, addressing areas that need enhancement and providing design documents for solutions.

## Use Cases

**Building Utilities with Your Machine:** Node.js can be used to create various utilities that automate tasks on your machine. This can include tasks related to managing files, performing automated backups, or even implementing live-reloading functionality for web development.

Sample Code (Live Reload Utility)

// Example of a live reload utility using Node.js

**const fs = require('fs');**

**const http = require('http');**

**fs.watch('path\_to\_watch\_directory', (event, filename) => {**

**if (event === 'change' && filename.endsWith('.html')) {**

**console.log(`Reloading ${filename}...`);**

// Implement code to trigger live reload in your application

**}**

**});**

**http.createServer((req, res) => {**

**res.write('Live Reload Utility is running!');**

**res.end();**

**}).listen(3000);**

**Web Application with a Web Server:** Node.js is commonly used to build web applications. It can serve as the backend server that handles HTTP requests and communicates with databases. Popular frameworks like Express.js provide a powerful platform for building web applications in Node.js.

Sample Code (Simple Webserver)

**var http = require(‘http’);**

**var server = http.createServer(function(request, response) {**

**console.log(‘Got a request!’);**

**response.write(‘hi’);**

**response.end();**

**});**

**Server.listen(3000);**

## Modules

**How to Load One File into Another:** Node.js allows you to modularize your code by breaking it into smaller, reusable files. You can use the **require** function to load these files into other files.

Sample Code (Loading a Module):

// File: myModule.js

**module.exports = () => {**

**console.log('This is my module!');**

**};**

// File: main.js

**const myModule = require('./myModule');**

**myModule();**

NPM Modules for Managing Packages: npm (Node Package Manager) is a tool that comes with Node.js, allowing you to install and manage external packages (libraries) for your projects. These packages can provide various functionalities and help you avoid reinventing the wheel.

// In your terminal, run:

**npm install packageName**

// In your code:

**const packageName = require('packageName');**

**Node Init for Managing Dependencies:** npm init is a command that helps you initialize a new Node.js project. It creates a package.json file which includes information about your project and its dependencies. This makes it easier to manage dependencies.

// In your terminal, run:

**npm init**

// Follow the prompts to create your package.json file

**Keeping Dependencies Updated:** npm allows you to easily update your project's dependencies to their latest versions. This ensures you're using the most up-to-date and secure packages.

// In your terminal, run:

**npm update**

// This will update your project's dependencies to their latest compatible versions

## Quick Quiz

1. What is Node JS? **Node.js is an open source server environment**
2. How is Node JS initiated on a computer? **Through the command line interface**
3. Why do we use Node JS? **Node JS is asynchronous**
4. What can Node JS do? **Node JS can send dynamic content   
   Node JS contains some tasks that can be executed on certain events eg someone trying to access a port on the server**
5. What is a module in Node JS the same as in JavaScript? **Libraries.**
6. What is NPM? **Node JS Package Manager**
7. What is contained  in a Node JS Package? **A package in Node.js contains all the files you need for a module**

# NodeJS Libraries

## Setup Environment

Steps to configure Environment and dependencies.

**npm init**

**npm install express**

**npm install nodemon --save-dev**

**npm i chalk --save-optional**

**npm i --save body-parser cors express socket.io**

**npm audit fix**

Put in package.json

**"scripts": {**

**"build": "webpack --progress --colors",**

**"dev": "nodemon client/server.js --exec babel-node"**

**},**

Create .gitignore file to hide /node\_modules/ from being pushed to github

**npm install webpack --save-dev**

**npx webpack**

# **React**

## What is React Native?

React Native is an open-source framework for building mobile applications. It's developed by Facebook and allows developers to use React (a popular JavaScript library for building user interfaces) to create native mobile apps for iOS and Android platforms.

Not HTML or webview

## Installing dependencies and setup server

# Create a new directory (or can make a project manually and use terminal in vs code)

**mkdir realtime-chat-app**

# Move into the new directory

**cd realtime-chat-app**

# Initialize a new package and install app dependencies

**npm init -y**

install specific packages

**npm install react react-dom next pusher pusher-js sentiment express body-parser cors dotenv axios**

**npm install express dotenv body-parser @pusher/chatkit-server cors date-fns @sendgrid/mail --save**

**OR** use this to install the app

**npx create-next-app@12 (name-of-project)**

# Move into the new directory

**cd realtime-chat-app**

npm install dotenv --save

npm install

npm run dev

## Create Pusher Account (Plus other chats)

<https://dashboard.pusher.com>

PUSHER\_APP\_ID=YOUR\_APP\_ID

PUSHER\_APP\_KEY=YOUR\_APP\_KEY

PUSHER\_APP\_SECRET=YOUR\_APP\_SECRET

PUSHER\_APP\_CLUSTER=YOUR\_APP\_CLUSTER

1. Create App with Pusher to generate keys
2. Create .env file in project and insert keys

[https://console.firebase.google.com/u/0/?\_gl=1\*5au76f\*\_ga\*MTU1ODAyNTAzNC4xNjk3NTQyMTA2\*\_ga\_CW55HF8NVT\*MTY5NzU0MjEwNy4xLjAuMTY5NzU0MjExMC41Ny4wLjA.&pli=1](https://console.firebase.google.com/u/0/?_gl=1*5au76f*_ga*MTU1ODAyNTAzNC4xNjk3NTQyMTA2*_ga_CW55HF8NVT*MTY5NzU0MjEwNy4xLjAuMTY5NzU0MjExMC41Ny4wLjA.&pli=1)

## Steps to building app

Dependencies

Modify the package.json file

Pusher account and generate keys

Create .env file (keys inside)

Create next.config.js

Create server.js

Create Layout.js

Create index.js

## Setting up the server

**/\* server.js \*/**

**const dotenv = require('dotenv').config();**

**const cors = require('cors');**

**const next = require('next');**

**const Pusher = require('pusher');**

**const express = require('express');**

**const bodyParser = require('body-parser');**

**const Sentiment = require('sentiment');**

**const dev = process.env.NODE\_ENV !== 'production';**

**const port = process.env.PORT || 3000;**

**const app = next({ dev });**

**const handler = app.getRequestHandler();**

**const sentiment = new Sentiment();**

**// Ensure that your pusher credentials are properly set in the .env file**

**// Using the specified variables**

**const pusher = new Pusher({**

**appId: process.env.PUSHER\_APP\_ID,**

**key: process.env.PUSHER\_APP\_KEY,**

**secret: process.env.PUSHER\_APP\_SECRET,**

**cluster: process.env.PUSHER\_APP\_CLUSTER,**

**encrypted: true**

**});**

**app.prepare()**

**.then(() => {**

**const server = express();**

**server.use(cors());**

**server.use(bodyParser.json());**

**server.use(bodyParser.urlencoded({ extended: true }));**

**server.get('\*', (req, res) => {**

**return handler(req, res);**

**});**

**server.listen(port, err => {**

**if (err) throw err;**

**console.log(`Ready on http://localhost:${port}`);**

**});**

**}**

**)**

**.catch(ex => {**

**console.error(ex.stack);**

**process.exit(1);**

**});**

## Modifying NPM Scripts

**/\* package.json \*/**

**"scripts": {**

**"dev": "node server.js",**

**"build": "next build",**

**"start": "NODE\_ENV=production node server.js"**

**}**