3/25/2022

Nagendra Mirajkar

Zensar AES Practices

Web API

Modern browser API’s

Table of Contents

[JavaScript Cookies 2](#_Toc99096208)

[What is a cookie 2](#_Toc99096209)

[Why cookies 2](#_Toc99096210)

[Cookie details 3](#_Toc99096211)

[Cookies in JavaScript 4](#_Toc99096212)

[**1) Get a cookie value** 4](#_Toc99096213)

[**2) Set a cookie** 4](#_Toc99096214)

[**3) Remove a cookie** 5](#_Toc99096215)

[JavaScript Cookie class 5](#_Toc99096216)

[**1) The get() method** 6](#_Toc99096217)

[**2) The set() method** 6](#_Toc99096218)

[**3) The remove() method** 6](#_Toc99096219)

[Using the Cookie class 7](#_Toc99096220)

[View cookies with web browsers 7](#_Toc99096221)

[Summary 7](#_Toc99096222)

[JavaScript localStorage 8](#_Toc99096223)

[Introduction to the Storage type 8](#_Toc99096224)

[The JavaScript localStorage object 8](#_Toc99096225)

[**localStorage vs. cookies** 8](#_Toc99096226)

[**Accessing the localStorage** 9](#_Toc99096227)

[**1) The setItem() method** 9](#_Toc99096228)

[**2) The length property** 9](#_Toc99096229)

[**3) The getItem() method** 9](#_Toc99096230)

[**4) The removeItem() method** 10](#_Toc99096231)

[**5) Loop over keys of the localStorage object** 10](#_Toc99096232)

[Storing objects 10](#_Toc99096233)

[The storage event 11](#_Toc99096234)

[Summary 11](#_Toc99096235)

[JavaScript sessionStorage 12](#_Toc99096236)

[Introduction to JavaScript sessionStorage 12](#_Toc99096237)

[Managing data in the JavaScript sessionStorage 13](#_Toc99096238)

[**1) Accessing the sessionStorage** 13](#_Toc99096239)

[**2) Storing data in the sessionStorage** 13](#_Toc99096240)

[**3) Getting data from the sessionStorage** 13](#_Toc99096241)

[**4) Removing an item by a name** 13](#_Toc99096242)

[**5) Iterating over all items** 13](#_Toc99096243)

[**6) Deleting all items in the sessionStorage** 14](#_Toc99096244)

[Why JavaScript sessionStorage 14](#_Toc99096245)

[JavaScript sessionStorage application 14](#_Toc99096246)

[**1) Creating the project folder structure** 14](#_Toc99096247)

[**2) Building the HTML page** 15](#_Toc99096248)

[**3) Creating app.js file** 15](#_Toc99096249)

[Summary 19](#_Toc99096250)

[JavaScript IndexedDB 19](#_Toc99096251)

[What is indexedDB 19](#_Toc99096252)

[Why indexedDB 19](#_Toc99096253)

[IndexedDB structure 20](#_Toc99096254)

[**Databases** 20](#_Toc99096255)

[**Object stores** 20](#_Toc99096256)

[**Indexes** 21](#_Toc99096257)

[Basic IndexedDB concepts 21](#_Toc99096258)

[**1) IndexedDB databases store key-value pairs** 21](#_Toc99096259)

[**2) IndexedDB is transactional** 21](#_Toc99096260)

[**3) IndexedDB API is mostly asynchronous** 21](#_Toc99096261)

[**4) IndexedDB is a NoSQL system** 21](#_Toc99096262)

[**5) IndexedDB follows the same-origin policy** 22](#_Toc99096263)

[Basic IndexedDB operations 22](#_Toc99096264)

[**1) Create the project structure** 22](#_Toc99096265)

[**1) Check if the IndexedDB is supported** 23](#_Toc99096266)

[**2) Open a database** 23](#_Toc99096267)

[**3) Create object stores** 24](#_Toc99096268)

[**4) Insert data into object stores** 25](#_Toc99096269)

[**5) Read data from the object store by key** 27](#_Toc99096270)

[**6) Read data from the object store by an index** 28](#_Toc99096271)

[**7) Read all data from an object store** 29](#_Toc99096272)

[**8) Delete a contact** 30](#_Toc99096273)

[**Put it all together** 31](#_Toc99096274)

[Summary 35](#_Toc99096275)

[JavaScript FormData 35](#_Toc99096276)

[Introduction to the JavaScript FormData API 35](#_Toc99096277)

[Create a FormData object 36](#_Toc99096278)

[FormData methods 37](#_Toc99096279)

[**append() method** 37](#_Toc99096280)

[**delete() method** 37](#_Toc99096281)

[**entries() method** 37](#_Toc99096282)

[**get() method** 37](#_Toc99096283)

[**getAll() method** 37](#_Toc99096284)

[**has() method** 38](#_Toc99096285)

[**keys() method** 38](#_Toc99096286)

[**set() method** 38](#_Toc99096287)

[**values() method** 38](#_Toc99096288)

[**Submit the FormData using fetch API** 38](#_Toc99096289)

[Put it all together. 40](#_Toc99096290)

[Summary 41](#_Toc99096291)

[JavaScript Drag and Drop 41](#_Toc99096292)

[Introduction to JavaScript Drag and Drop API 41](#_Toc99096293)

[Events on draggable elements 42](#_Toc99096294)

[Events on drop targets 42](#_Toc99096295)

[Valid drop target 43](#_Toc99096296)

[Transfer data using dataTransfer object 43](#_Toc99096297)

[JavaScript drag and drop example 44](#_Toc99096298)

[**Create the project structure** 44](#_Toc99096299)

[**Build the index.html file** 44](#_Toc99096300)

[**Handle events on the draggable element** 46](#_Toc99096301)

[**Handle events on drop targets** 48](#_Toc99096302)

[Summary 52](#_Toc99096303)

[JavaScript FileReader 52](#_Toc99096304)

[Introduction to the JavaScript FileReader API 52](#_Toc99096305)

[Using JavaScript FileReader to implement an image upload application 53](#_Toc99096306)

[**Setting up the project structure** 54](#_Toc99096307)

[**index.html** 55](#_Toc99096308)

[**app.js** 56](#_Toc99096309)

[Summary 66](#_Toc99096310)

[JavaScript Geolocation 66](#_Toc99096311)

[What is Geolocation API 66](#_Toc99096312)

[Why Geolocation API 66](#_Toc99096313)

[The geolocation object 66](#_Toc99096314)

[**Check if the Geolocation API is supported** 67](#_Toc99096315)

[**Get the current position** 67](#_Toc99096316)

[JavaScript Geolocation example 67](#_Toc99096317)

[Test the app 69](#_Toc99096318)

[Summary 72](#_Toc99096319)

[JavaScript Notification 72](#_Toc99096320)

[Notification permissions 72](#_Toc99096321)

[Show and hide notifications 73](#_Toc99096322)

[Notification events 73](#_Toc99096323)

[JavaScript Notification API example 74](#_Toc99096324)

[**Create the project structure** 74](#_Toc99096325)

[**Create the index.html page** 75](#_Toc99096326)

[**Make the app.js file** 75](#_Toc99096327)

[Summary 78](#_Toc99096328)

[JavaScript Canvas 79](#_Toc99096329)

[Introduction to the HTML5 Canvas element 79](#_Toc99096330)

[**Fallback content** 79](#_Toc99096331)

[**The rendering context** 80](#_Toc99096332)

[**Check for canvas support** 80](#_Toc99096333)

[The 2D context 80](#_Toc99096334)

[**Fills and strokes** 81](#_Toc99096335)

[Summary 81](#_Toc99096336)

[JavaScript fillRect 81](#_Toc99096337)

[Introduction to the JavaScript fillRect() method 81](#_Toc99096338)

[JavaScript fillRect() example 82](#_Toc99096339)

[Using the JavaScript fillRect() to draw a brick wall 84](#_Toc99096340)

[Summary 87](#_Toc99096341)

[JavaScript strokeRect 87](#_Toc99096342)

[Introduction to the JavaScript strokeRect() method 87](#_Toc99096343)

[The JavaScript strokeRect() example 88](#_Toc99096344)

[Summary 90](#_Toc99096345)

[Canvas clearRect 90](#_Toc99096346)

[Introduction to the Canvas clearRect() method 90](#_Toc99096347)

[The canvas clearRect() method example 91](#_Toc99096348)

[Summary 93](#_Toc99096349)

[How to Draw a Line in JavaScript 93](#_Toc99096350)

[Steps for drawing a line in JavaScript 93](#_Toc99096351)

[**Set the line stroke** 93](#_Toc99096352)

[**Set the line width** 93](#_Toc99096353)

[**The lineTo(x,y) method** 93](#_Toc99096354)

[Drawing a line example 94](#_Toc99096355)

[**Develop a resuable drawLine() function** 95](#_Toc99096356)

[Summary 96](#_Toc99096357)

[JavaScript fillText 96](#_Toc99096358)

[Introduction to the JavaScript fillText() method 96](#_Toc99096359)

[JavaScript fillText() example 97](#_Toc99096360)

[**1) Draw a filled text example** 97](#_Toc99096361)

[**2) Constraint the text size** 98](#_Toc99096362)

[Text alignments 99](#_Toc99096363)

[Text baseline 101](#_Toc99096364)

[**Text baseline example** 101](#_Toc99096365)

[**JavaScript** 102](#_Toc99096366)

[Summary 103](#_Toc99096367)

[JavaScript Translate 103](#_Toc99096368)

[Introduction to the JavaScript translate() method. 104](#_Toc99096369)

[JavaScript translate() examples 105](#_Toc99096370)

[**1) Simple JavaScript translate() example** 105](#_Toc99096371)

[**JavaScript** 106](#_Toc99096372)

[**2) Using JavaScript translate() to draw a clock** 106](#_Toc99096373)

[Summary 107](#_Toc99096374)

[JavaScript Rotate 107](#_Toc99096375)

[Introduction to JavaScript rotate() canvas API 108](#_Toc99096376)

[JavaScript rotate() example 109](#_Toc99096377)

[**HTML** 109](#_Toc99096378)

[**JavaScript** 109](#_Toc99096379)

[Summary 110](#_Toc99096380)

[JavaScript Scale 111](#_Toc99096381)

[Introduction to JavaScript scale() Canvas API 111](#_Toc99096382)

[JavaScript scale() examples 111](#_Toc99096383)

[**HTML** 111](#_Toc99096384)

[**JavaScript** 111](#_Toc99096385)

[Summary 112](#_Toc99096386)

[JavaScript History pushState 113](#_Toc99096387)

[Introduction to the JavaScript history pushState() method 113](#_Toc99096388)

[**1) state** 113](#_Toc99096389)

[**2) title** 113](#_Toc99096390)

[**3) url** 113](#_Toc99096391)

[JavaScript history pushState() example 113](#_Toc99096392)

[**Make the index.html page** 114](#_Toc99096393)

[Summary 119](#_Toc99096394)

[JavaScript Fetch API 119](#_Toc99096395)

[Sending a Request 119](#_Toc99096396)

[Reading the Response 120](#_Toc99096397)

[Handling the status codes of the Response 120](#_Toc99096398)

[JavaScript Fetch API example 121](#_Toc99096399)

[Summary 123](#_Toc99096400)

This section introduces you to the Web APIs specified in the HTML5 specification. These Web APIs allow you to use the modern functions provided by the web browsers in your web applications.

JavaScript Cookies

**Summary**: in this tutorial, you’ll learn about HTTP cookies and how to use JavaScript to manage the cookies effectively.

## **What is a cookie**

An HTTP cookie is a piece of data that a server sends to a web browser. Then, the web browser stores the HTTP cookie on the user’s computer and sends it back to the same server in the later requests.

An HTTP cookie is also known as a web cookie or browser cookie. And it is commonly called a cookie.

For example, the header of an HTTP response may look like this:

HTTP/1.1 200 OK

Content-type: text/html

Set-Cookie:username=admin

...

Code language: JavaScript (javascript)

The HTTP response sets a cookie with the name of "username" and value of "admin". The server encodes both name and value when sending the cookie to the web browser.

The web browser stores this information and sends it back to the server via the Cookie HTTP header for the next request as follows:

GET /index.html HTTP/1.1

Cookie: username=admin

...

## **Why cookies**

It’s important to understand that **HTTP is stateless**. When you send two subsequent HTTP requests to the server, there is no link between them. In other words, the server cannot know if the two requests are from the same web browser.

Therefore, a cookie is used to tell if the two requests came from the same web browser.

In practice, cookies serve the following purposes:

* Session management – cookies allow you to manage any information that the server should remember. For example, logins, shopping carts, etc.
* Personalization – cookies allow you to store user preferences, themes, and setting specific to a user.
* Tracking – cookies help record and analyze user behaviors in advertising.

## **Cookie details**

A cookie consists of the following information stored by the web browser:

* Name – a unique name that identifies the cookie. The cookie names are case-insensitive. It means that Username and username are the same cookies.
* Value – string value of the cookie. It must be URL-encoded.
* Domain – a domain for which the cookie is valid.
* Path – path without the domain for which the cookie should be sent to the server. For example, you can specify that the cookie is accessible only from the https://www.javascripttutorial.net/dom/ so pages at https://www.javascripttutoiral.net/ won’t send the cookie information.
* Expiration – timestamp that indicates when the web browser should delete the cookie (or when the browser should stop sending the cookie to the server). The expiration date is set as a date in GMT format: Wdy, DD-Mon-YYYY HH:MM:SS GMT. The expiration date allows the cookies to be stored in the user’s web browsers even after users close the web browsers.
* Secure flag – if specified, the web browser only sends the cookie to the server only via an SSL connection (https, not http)

The name, value, domain, path, expiration, and secure flag are separated by a semicolon and space. For example:

HTTP/1.1 200 OK

Content-type: text/html

Set-Cookie:user=john

; expire=Tue, 12-December-2030 12:10:00 GMT; domain=javascripttutorial.net; path=/dom; secure

...

Note that the secure flag is the only part that is not a name-value pair.

## **Cookies in JavaScript**

To manage cookies in JavaScript, you use the document.cookie property.

### **1) Get a cookie value**

The following example returns a string of all cookies available to the page:

const str = document.cookie;

Code language: JavaScript (javascript)

The document.cookie returns a series of name-value pairs separated by semicolons like this:

name1=value1;name2=value2;...

The following example shows the cookies of a webpage:

"\_ga=GA1.2.336374160.1600215156; dwf\_sg\_task\_completion=False; \_gid=GA1.2.33408724.1600901684"

Code language: JSON / JSON with Comments (json)

Since all the names and values are URL-encoded, you need to use the decodeURIComponent() to decode them.

### **2) Set a cookie**

To set a value for a cookie, you compose the cookie text in the following format:

name=value; expires=expirationTime; path=domainPath; domain=domainName; secure

…and append it to the cookie:

document.cookie = cookieText;

Code language: JavaScript (javascript)

A cookie requires only name and value. For example:

document.cookie = "username=admin";

Code language: JavaScript (javascript)

This example creates a cookie called username that has a value of admin. The web browser will send this cookie every time it makes a request to the server.

Since the cookie doesn’t specify the expired time, it will be deleted when the web browser is closed.

The cookie text "username=admin" doesn’t have any character that needs to be encoded.

However, it’s a good practice to always use the encodeURIComponent() function when setting a cookie like this:

document.cookie = `${encodeURIComponent("username")}=${encodeURIComponent("admin")}`;

Code language: JavaScript (javascript)

### **3) Remove a cookie**

To remove a cookie, you need to set the cookie again with the same name, path, domain, and secure option. And you need to set the expiration date to some time in the past.

## **JavaScript Cookie class**

The following Cookie class sets, gets, and removes a cookie:

class Cookie {

static get(name) {

const cookieName = `${encodeURIComponent(name)}=`;

const cookie = document.cookie;

let value = null;

const startIndex = cookie.indexOf(cookieName);

if (startIndex > -1) {

const endIndex = cookie.indexOf(';', startIndex);

if (endIndex == -1) {

endIndex = cookie.length;

}

value = decodeURIComponent(

cookie.substring(startIndex + name.length, endIndex)

);

}

return value;

}

static set(name, value, expires, path, domain, secure) {

let cookieText = `${encodeURIComponent(name)}=${encodeURIComponent(value)}`;

if (expires instanceof Date) {

cookieText += `; expires=${expires.toGMTString()}`;

}

if (path) cookieText += `; path=${path}`;

if (domain) cookieText += `; domain=${domain}`;

if (secure) cookieText += `; secure`;

document.cookie = cookieText;

}

static remove(name, path, domain, secure) {

Cookie.set(name, '', new Date(0), path, domain, secure);

}

}

Code language: JavaScript (javascript)

How it works.

The Cookie class has three static methods: get(), set(), and remove().

### **1) The get() method**

The get() method returns the value of a cookie with a specified name. To do so, it performs the following steps:

* First, find the occurrence of the cookie name by an equal sign in the document.cookie property.
* Second, if the cookie is available, it uses the indexOf() to find the end of the cookie, which is specified by the next semicolon (;) after that location. If the semicolon isn’t available, this means that the cookie is the last one in the string.
* Third, decode the value of the cookie using the decodeURIComponent() function and return the decoded value.

### **2) The set() method**

The set() method sets a cookie on the page. It accepts the arguments required to construct a cookie.

The set() method requires the first two arguments: name and value. The other arguments aren’t mandatory.

The set() method composes a cookie text and sets it to the document.cookie property.

### **3) The remove() method**

To remove a cookie, the remove() method sets the cookie again with the expiration date set to January 1, 1970. Note that the new Date(0) returns a date object whose date is January 1, 1970.

## **Using the Cookie class**

The following shows how to use the Cookie class to set, get, and remove a cookie whose name is username and value is admin:

*// set a cookie*

Cookie.set('username', 'admin');

*// get a cookie*

console.log(Cookie.get('username')); *// admin*

*// remove a cookie by a name*

Cookie.remove('username');

Code language: JavaScript (javascript)

## **View cookies with web browsers**

To view the cookies on the web browser, you use the devtools.

* First, click the application tab.
* Second, select **Cookies**node under the **Storage**.

The following picture shows the cookies of Google.com:

Graphical user interface, text, application

Description automatically generated

## **Summary**

* A cookie is a piece of data that a server sends to a web browser. The web browser then stores it in the user’s computer and sends the cookie back to the same server in the subsequent requests.
* The server uses cookies for identifying if two successive requests came from the same web browser.
* To manage cookies, you use the document.cookie object. To make it more efficient, you can use the Cookie utility class.
* Use the encodeURIComponent() and decodeURIComponent() function to encode and decode the cookie values.

# JavaScript localStorage

**Summary**: in this tutorial, you’ll learn about the Storage type and how to use the JavaScript localStorage to store persistent data.

## **Introduction to the Storage type**

The Storage type is designed to store name-value pairs. The Storage type is an Object with the following additional methods:

* setItem(name, value) – set the value for a name
* removeItem(name) – remove the name-value pair identified by name.
* getItem(name) – get the value for a given name.
* key(index) – get the name of the value in the given numeric position.
* clear() – remove all values.

To get the number of name-value pairs in a Storage object, you can use the length property.

The Storage object can store only strings. It’ll automatically convert non-string data into a string before storing it.

When you retrieve data from a Storage object, you’ll always get the string data.

## **The JavaScript localStorage object**

HTML5 specification introduces the localStorage as a way to store data with no expiration date in the web browsers.

In other words, the data stored in the browsers will persist even after you close the browser windows.

The data stored in the localStorage is bound to an origin. It means that the localStorage is unique per protocol://host:port.

### **localStorage vs. cookies**

First, the data stored in the localStorage isn’t sent to the server in every request like [cookies](https://www.javascripttutorial.net/web-apis/javascript-cookies/). For this reason, you can store more data in the localStorage.

Most modern web browsers allow you to store up to 5MB of data in the localStorage. Note that you can store up to 4KB in cookies.

Second, the data stored in the localStorage can be managed by the client, specifically JavaScript in the web browser. It cannot be accessible by the servers.

However, cookies can be managed by both JavaScript in web browsers and servers.

### **Accessing the localStorage**

You can access the localStorage via the property of the window object:

window.localStorage

Code language: JavaScript (javascript)

Since the localStorage is an instance of the Storage type, you can invoke the methods of the Storage type to manage data.

When you type the following code in the Console:

window.localStorage

Code language: JavaScript (javascript)

… you’lll see the following object:

Storage {length: 0}

Code language: CSS (css)

### **1) The setItem() method**

The following uses the setItem() method to store a name-value pair in the localStorage:

window.localStorage.setItem('theme','dark');

Code language: JavaScript (javascript)

### **2) The length property**

To get the number of name-value pairs, you use the length property like this:

console.log(window.localStorage.length); *// 1*

Code language: JavaScript (javascript)

Since the window object is [global](https://www.javascripttutorial.net/es-next/javascript-globalthis/), you don’t need to explicitly specify it. For example:

console.log(localStorage.length); *// 1*

Code language: JavaScript (javascript)

### **3) The getItem() method**

To get the value by a key, you use the getItem() method. The following example uses the getItem() method to get the value of theme key:

localStorage.getItem('theme'); *// 'dark'*

Code language: JavaScript (javascript)

### **4) The removeItem() method**

To remove a name-value pair by a key, you use the removeItem() method. For example:

localStorage.removeItem('theme');

Code language: JavaScript (javascript)

### **5) Loop over keys of the localStorage object**

The following stores three name-value pairs to the localStorage:

localStorage.setItem('theme','light');

localStorage.setItem('backgroundColor','white');

localStorage.setItem('color','#111');

Code language: JavaScript (javascript)

To iterate over name-value pairs stored in the localStorage, you use the [Object.keys()](https://www.javascripttutorial.net/object/iterate-object-in-javascript/) method with [for...of](https://www.javascripttutorial.net/es6/javascript-for-of/) loop:

let keys = Object.keys(localStorage);

for(let key of keys) {

console.log(`${key}: ${localStorage.getItem(key)}`);

}

Code language: JavaScript (javascript)

Output:

color: #111

theme: light

backgroundColor: white

Code language: HTTP (http)

## **Storing objects**

The Storage type stores only string data. To store objects, you need to convert them into strings using the JSON.stringify() method. For example:

const settings = {

backgroundColor: '#fff',

color: '#111',

theme: 'light'

};

localStorage.setItem('settings', JSON.stringify(settings));

console.log(localStorage.getItem('settings'));

Code language: JavaScript (javascript)

Output: (a string)

'{"backgroundColor":"#fff","color":"#111","theme":"light"}'

Code language: JavaScript (javascript)

The following retrieves the value from the localStorage and converts it back to the object using the JSON.parse() method.

let storedSettings = JSON.parse(localStorage.getItem('settings'));

console.log(storedSettings);

Code language: JavaScript (javascript)

## **The storage event**

When you make a change to the Storage object, the storage event is fired on the document.

The storage event occurs in the following scenarios:

* Store a name-value pair by calling the setItem() method.
* Remove a name-value pair by calling the removeItem() method.
* And remove all values by calling the clear() method.

The storage event has the following properties:

* domain – the domain which the storage changes for.
* key – the key that was set or removed.
* newValue – the value that the key was set to or null if the key was removed.
* oldValue – the value before the key was set or removed.

To listen for the storage event, you use the [addEventListener()](https://www.javascripttutorial.net/dom/events/add-an-event-handler/) method of the window object like this:

addEventListener('storage', function(e){

console.log(`The value of the ${e.key} changed for the ${e.domain}.`);

});

Code language: JavaScript (javascript)

## **Summary**

* The Storage type provides you with the methods for storing and managing data in the web browsers.
* The localStorage is an instance of the Storage type that allows you to store persistent data in the web browsers.
* The localStorage can store only strings. To store objects, you convert them to strings using the JSON.stringify() method. And you convert the strings into objects when you retrieve them from the localStorage using the JSON.parse() method.

# JavaScript sessionStorage

**Summary**: in this tutorial, you’ll learn how to use the JavaScript sessionStorage to store data only for a session.

## **Introduction to JavaScript sessionStorage**

The sessionStorage object stores data only for a session. It means that the data stored in the sessionStorage will be deleted when the browser is closed.

A page session lasts as long as the web browser is open and survives over the page refresh.

When you open a page in a new tab or window, the web browser creates a new session.

If you open multiple tabs or windows with the same URL, the web browser creates a separate sessionStorage for each tab or window. So data stored in one web browser tab cannot be accessible in another tab.

When you close a tab or window, the web browser ends the session and clears data in the sessionStorage.

Data stored in the sessionStorage is specific to the protocol of the page. For example, the same site javascripttutorial.net has different sessionStorage when accessing with the http and https.

Since the sessionStorage data is tied to a server session, it’s only available when a page is requested from a server. The sessionStorage isn’t available when the page runs locally without a server.

Because the sessionStorage is an instance of the [Storage](https://www.javascripttutorial.net/web-apis/javascript-localstorage/) type, you can manage data using the Storage’s methods:

* setItem(name, value) – set the value for a name
* removeItem(name) – remove the name-value pair identified by name.
* getItem(name) – get the value for a given name.
* key(index) – get the name of the value in the given numeric position.
* clear() – remove all values in the sessionStorage .

## **Managing data in the JavaScript sessionStorage**

### **1) Accessing the sessionStorage**

To access the sessionStorage, you use the sessionStorage property of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object:

window.sessionStorage

Code language: JavaScript (javascript)

Since the window is the [global object](https://www.javascripttutorial.net/es-next/javascript-globalthis/), you can simply access the sessionStorage like this:

sessionStorage

### **2) Storing data in the sessionStorage**

The following stores a name-value pair in the sessionStorage:

sessionStorage.setItem('mode','dark');

Code language: JavaScript (javascript)

If the sessionStorage has an item with the name of mode, the setItem() method will update the value for the existing item to dark. Otherwise, it’ll insert a new item.

### **3) Getting data from the sessionStorage**

To get the value of an item by name, you use the getItem() method. The following example gets the value of the item ‘mode‘:

const mode = sessionStorage.getItem('mode');

console.log(mode); *// 'dark'*

Code language: JavaScript (javascript)

If there is no item with the name mode, the getItem() method will return null.

### **4) Removing an item by a name**

To remove an item by the name, you use the removeItem() method. The following removes the item with the name of 'mode':

sessionStorage.removeItem('mode');

Code language: JavaScript (javascript)

### **5) Iterating over all items**

To iterate over all items stored in the sessionStorage, you follow these steps:

* Use Object.keys() to get all keys of the sessionStorage object.
* Use [for...of](https://www.javascripttutorial.net/es6/javascript-for-of/) to iterate over the keys and get the items by keys.

The following code illustrates the steps:

let keys = Object.keys(sessionStorage);

for(let key of keys) {

console.log(`${key}: ${sessionStorage.getItem(key)}`);

}

Code language: JavaScript (javascript)

### **6) Deleting all items in the sessionStorage**

The data stored in the sessionStorage are automatically deleted when the web browser tab/window is closed.

In addition, you can use the clear() method to programmatically delete all data stored in the sessionStorage.

sessionStorage.clear();

Code language: CSS (css)

## **Why JavaScript sessionStorage**

The sessionStorage has many practical applications. And the following are the notable ones:j

* The sessionStorage can be used to store the state of the user interface of the web application. Later, when the user comes back to the page, you can restore the user interface stored in the sessionStorage.
* The sessionStorage can also be used to pass data between pages instead of using the hidden input fields or URL parameters.

## **JavaScript sessionStorage application**

You’ll build a simple web application that allows users to select the mode, either dark or light mode. By default, it has a light mode. And you’ll use the sessionStorage to remember the mode when the page refreshes.

If you refresh the page, the mode that you selected will restore since it’s stored in the sessionStorage.

However, if you close the tab or window, the page will reset to the dark mode, which is the default mode.

### **1) Creating the project folder structure**

First, create a new folder called session-storage. In the session-storage folder, create two subfolders: js and css that will store the JavaScript and CSS files.

Second, create a new index.html in the sessionStorage folder, the app.js file in the js folder, and style.css file in the css folder.

### **2) Building the HTML page**

The following shows the index.html page:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript sessionStorage Demo**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<div class="container">**

**<h1>**JavaScript sessionStorage Demo**</h1>**

**<p>**Click the button to switch to the dark/light mode.**</p>**

**<p>**Refresh the page to check if the mode is saved.**</p>**

**<a id="theme-switcher" class="btn"></a>**

**</div>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this index.html file, we place the style.css in the head section and app.js in the body section.

The page has some elements. The most important one is the button with the id theme-switcher.

### **3) Creating app.js file**

First, declare two constants that will be used as the butotn’s label:

const MOON = '🌙';

const SUN = '☀️';

Code language: JavaScript (javascript)

You’ll use the SUN as the label of the theme-switcher button in the dark mode and MOON in the light mode.

Second, declare three constants for the dark, light, and default modes:

const DARK\_MODE = 'dark';

const LIGHT\_MODE = 'light';

const DEFAULT\_MODE = DARK\_MODE;

Code language: JavaScript (javascript)

Third, select the button theme-switcher by using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/):

const btn = document.querySelector('#theme-switcher');

Code language: JavaScript (javascript)

Fourth, define a new function setMode() to change the mode:

function setMode(mode = DEFAULT\_MODE) {

if (mode === DARK\_MODE) {

btn.textContent = SUN;

document.body.classList.add(DARK\_MODE);

} else if (mode === LIGHT\_MODE) {

btn.textContent = MOON;

document.body.classList.remove(DARK\_MODE);

}

}

Code language: JavaScript (javascript)

In the dark mode, the setMode() changes the button to SUN and adds the DARK\_MODE class to the body element

And in the light mode, the setMode() changes the button label to MOON and removes the DARK\_MODE class from the body element.

The following shows the CSS of the light mode. The background color is white and the text color is black:

body {

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;

font-size: 16px;

background-color: #fff;

color: #333;

line-height: 1.7;

transition: 0.2s ease-in-out;

padding: 20px;

}

Code language: CSS (css)

In the dark mode, the background color is black and the text color is white:

.dark {

background-color: black;

color: #fff;

}

Code language: CSS (css)

To switch from the light to dark mode, you add the .dark class to the body element and vice versa.

Fifth, define the init() function that will run when the page loads:

function init() {

let storedMode = sessionStorage.getItem('mode');

if (!storedMode) {

storedMode = DEFAULT\_MODE;

sessionStorage.setItem('mode', DEFAULT\_MODE);

}

setMode(storedMode);

}

Code language: JavaScript (javascript)

In this function, we use the getItem() method to retrieve the mode stored in the sessionStorage.

If the sessionStorage doesn’t have the mode item, the init() function will switch the page to the default mode, which is the dark mode. Otherwise, it sets to the mode stored in the sessionStorage.

Sixth, attach a click event handler to the theme-switcher button:

btn.addEventListener('click', function () {

let mode = sessionStorage.getItem('mode');

if (mode) {

let newMode = mode == DARK\_MODE ? LIGHT\_MODE : DARK\_MODE;

setMode(newMode);

sessionStorage.setItem('mode', newMode);

}

});

Code language: JavaScript (javascript)

The click event handler gets the mode stored in the sessionStorage.

If the mode item exists, it toggles the mode. In other words, the light mode becomes the dark mode and vice versa.

It then uses the setItem() method to update the mode item in the sessionStorage to the new one.

The following shows a complete app.js file:

const MOON = '🌙';

const SUN = '☀️';

const DARK\_MODE = 'dark';

const LIGHT\_MODE = 'light';

const DEFAULT\_MODE = DARK\_MODE;

const btn = document.querySelector('#theme-switcher');

init();

function init() {

let storedMode = sessionStorage.getItem('mode');

if (!storedMode) {

storedMode = DEFAULT\_MODE;

sessionStorage.setItem('mode', DEFAULT\_MODE);

}

setMode(storedMode);

}

function setMode(mode = DEFAULT\_MODE) {

if (mode === DARK\_MODE) {

btn.textContent = SUN;

document.body.classList.add(DARK\_MODE);

} else if (mode === LIGHT\_MODE) {

btn.textContent = MOON;

document.body.classList.remove(DARK\_MODE);

}

}

btn.addEventListener('click', function () {

let mode = sessionStorage.getItem('mode');

if (mode) {

let newMode = mode == DARK\_MODE ? LIGHT\_MODE : DARK\_MODE;

setMode(newMode);

sessionStorage.setItem('mode', newMode);

}

});

Code language: JavaScript (javascript)

[Here is the final application](https://www.javascripttutorial.net/sample/webapis/session-storage/).

First, you select a mode e.g., light mode, the sessionStorage will save it.

Then, you refresh the page. It’ll show the previously selected mode.

To view the data stored in the session storage in the web browser, you click the Application tab and select the Session Storage:

Graphical user interface, application

Description automatically generated

## **Summary**

* The sessionStorage allows you to store the data for session only. The browser will delete the sessionStorage data when you close the browser tab or window.
* The sessionStorage is an instance of the Storage type, therefore, you can use the methods of the Storage type to manage data in the sessionStorage.

# JavaScript IndexedDB

**Summary**: in this tutorial, you’ll learn about the IndexedDB and how to use it to persistently store data inside the browser.

## **What is indexedDB**

IndexedDB is a **large-scale object store** built into the browser.

The IndexedDB allows you to persistently store the data using key-value pairs.

The values can be any [JavaScript type](https://www.javascripttutorial.net/javascript-data-types/) including [boolean](https://www.javascripttutorial.net/javascript-boolean/), [number](https://www.javascripttutorial.net/javascript-number/), [string](https://www.javascripttutorial.net/javascript-string-type/), [undefined](https://www.javascripttutorial.net/javascript-undefined/), null, date,[object](https://www.javascripttutorial.net/javascript-objects/), [array](https://www.javascripttutorial.net/javascript-array/), [regex](https://www.javascripttutorial.net/javascript-regular-expression/), blob, and files.

## **Why indexedDB**

IndexedDB allows you to create web applications that can work both online and offline.

It’s useful for applications that store a large amount of data and don’t need a persistent internet connection.

For example, Google Docs uses the IndexedDB to store the cached documents in the browser and synchronizes with the server once in a while. This allows Google Docs to increase performance while enhancing user experiences.

And you’ll find other types of applications that heavily use IndexedDB like online notepads, quizzes, todo lists, code sandboxes, and CMS.

## **IndexedDB structure**

The following picture illustrates the structure of the IndexedDB:

Chart

Description automatically generated

### **Databases**

A database is the highest level of IndexedDB. A database contains one or more object stores.

The IndexedDB can have one or more databases. Generally, you’ll create one database per web application.

### **Object stores**

An object store is a bucket that you can use to store the data and associated indexes. It’s conceptually equivalent to the tables in SQL databases.

An object store contains the records stored as key-value pairs.

### **Indexes**

Indexes allow you to query data by the [properties](https://www.javascripttutorial.net/javascript-object-properties/) of the objects.

Technically, you create indexes on object stores, which are called parent object stores.

For example, if you store the contact information, you may want to create indexes on email, first name, and last name so that you can query the contacts by these properties.

## **Basic IndexedDB concepts**

The following briefly introduces the basic concepts in the IndexedDB:

### **1) IndexedDB databases store key-value pairs**

Unlike [localStorage](https://www.javascripttutorial.net/web-apis/javascript-localstorage/) and [sessionStorage](https://www.javascripttutorial.net/web-apis/javascript-sessionstorage/), the values stored in the IndexedDB can be complex structures like objects and blob.

Also, keys can be the properties of these objects or can be binary objects.

For quick searching and sorting, you can create indexes that use any property of the objects.

### **2) IndexedDB is transactional**

Every read from and write to the IndexedDB databases always happens in a transaction.

The transactional model ensures the data integrity in case users open the web application in two tabs/windows at the same time and perform the read from and write to the same database.

### **3) IndexedDB API is mostly asynchronous**

IndexedDB operations are asynchronous. It uses DOM events to notify you when an operation completes and the result is available.

### **4) IndexedDB is a NoSQL system**

The IndexedDB is a NoSQL system. In other words, it doesn’t use SQL to query data. Instead, it uses the query that returns a cursor. Then, you can use the cursor to iterate the result set.

### **5) IndexedDB follows the same-origin policy**

An origin is a domain, protocol, and port of a URL of the document where the code executes. For example https://www.javascripttutorial.net:

* domain: javascripttutorial.net
* protocol: https
* port: 443

The https://www.javascripttutorial.net/dom/ and https://www.javascripttutorial.net/ are the same-origin because they have the same domain, protocol, and port.

However, the https://www.javascripttutorial.net/ and https://www.javascripttutorial.net/ aren’t the same origin since they have different protocol and port:

|  |  |  |
| --- | --- | --- |
|  | https://www.javascripttutorial.net | http://www.javascripttutorial.net |
| Protocol | https | http |
| Port | 443 | 80 |

IndexedDB adheres to the same-origin policy. It means that each origin has its own set of databases. And one origin cannot access databases from other origins.

## **Basic IndexedDB operations**

The following describes the basic operations on the IndexedDB databases such as

* Opening a connection to a database.
* Inserting an object into the object store.
* Reading data from the object store.
* Using a cursor to iterate over a result set.
* Deleting an object from the object store.

Before opening a connection to a database in the IndexedDB, let’s create the project structure first.

### **1) Create the project structure**

First, create a new folder called indexeddb folder. Inside the indexeddb folder, create another subfolder called js.

Second, create the index.html in the indexeddb folder, app.js in the js folder.

Third, place the <script> tag that links to the app.js file in the index.html file like this:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**IndexedDB**</title>**

**</head>**

**<body>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In the app.js, you’ll place all the JavaScript code an [IIFE](https://www.javascripttutorial.net/javascript-immediately-invoked-function-expression-iife/).

(function () {

*// all the code will be here*

*// ...*

})();

Code language: JavaScript (javascript)

### **1) Check if the IndexedDB is supported**

The following code checks if a web browser supports the IndexedDB:

if (!window.indexedDB) {

console.log(`Your browser doesn't support IndexedDB`);

return;

}

Code language: JavaScript (javascript)

Since most modern web browsers support the IndexedDB, this may not be necessary anymore.

### **2) Open a database**

To open a connection to a database, you use the open() method of the window.indexedDB:

const request = indexedDB.open('CRM', 1);

Code language: JavaScript (javascript)

The open() method accepts two arguments:

* The database name (CRM)
* The database version (1)

The open() method returns a request object which is an instance of the IDBOpenDBRequest interface.

When you call the open() method, it can succeed or fail. To handle each case, you can assign the corresponding event handler as follows:

request.onerror = (event) => {

console.error(`Database error: ${event.target.errorCode}`);

};

request.onsuccess = (event) => {

*// add implementation here*

};

Code language: JavaScript (javascript)

### **3) Create object stores**

When you open the database for the first time, the onupgradeneeded event will trigger.

If you open the database from the second time with a version higher than the existing version, the onupgradeneeded  event also triggers.

At the first time, you can use the onupgradeneeded event handler to initialize the object stores and indexes.

For example, the following onupgradeneeded event handler creates the Contacts object store and its index.

*// create the Contacts object store and indexes*

request.onupgradeneeded = (event) => {

let db = event.target.result;

*// create the Contacts object store*

*// with auto-increment id*

let store = db.createObjectStore('Contacts', {

autoIncrement: true

});

*// create an index on the email property*

let index = store.createIndex('email', 'email', {

unique: true

});

};

Code language: JavaScript (javascript)

How it works.

* First, get the IDBDatabase instance from the event.target.result and assign it to the db variable.
* Second, call the createObjectStore() method to create the Contacts object store with the autoincrement key. It means that the IndexedDB will generate an auto-increment number starting at one as the key for every new object inserted into the Contacts object store.
* Third, call the createIndex() method to create an index on the email property. Since the email is unique, the index should also be unique. To do so, you specify the third argument of the createIndex() method { unique: true }.

### **4) Insert data into object stores**

Once you open a connection to the database successfully, you can manage data in the onsuccess event handler.

For example, to add an object to an object store, you follow these steps:

* First, open a new transaction.
* Second, get an object store.
* Third, call the put() method of the object store to insert a new record.
* Finally, close the connection to the database once the transaction completes.

The following insertContact() function inserts a new contact into the Contacts object store:

function insertContact(db, contact) {

*// create a new transaction*

const txn = db.transaction('Contacts', 'readwrite');

*// get the Contacts object store*

const store = txn.objectStore('Contacts');

*//*

let query = store.put(contact);

*// handle success case*

query.onsuccess = function (event) {

console.log(event);

};

*// handle the error case*

query.onerror = function (event) {

console.log(event.target.errorCode);

}

*// close the database once the*

*// transaction completes*

txn.oncomplete = function () {

db.close();

};

}

Code language: JavaScript (javascript)

To create a new transaction, you call the transaction() method of the IDBDatabase object.

You can open a transaction in one of two modes: readwrite or readonly. The readwrite mode allows you to read data from and write data to the database while the readonly mode allows you to only read data from the database.

It’s a good practice to open a readonly transaction if you need to read data from a database only.

After defining the insertContact() function, you can call it in the onsuccess event handler of the request to insert one or more contacts like this:

request.onsuccess = (event) => {

const db = event.target.result;

insertContact(db, {

email: 'john.doe@outlook.com',

firstName: 'John',

lastName: 'Doe'

});

insertContact(db, {

email: 'jane.doe@gmail.com',

firstName: 'Jane',

lastName: 'Doe'

});

};

Code language: JavaScript (javascript)

Now, if you open the index.html file in the web browser, the code in the app.js will execute to:

* Create the CRM database in the IndexedDB.
* Create the Contacts object store in the CRM database.
* Insert two records into the object store.

If you open the devtools on the web browser, you’ll see the CRM database with the Contacts object store. And in the Contacts object store, you’ll see the data there as shown in the following picture:

Graphical user interface, application

Description automatically generated

### **5) Read data from the object store by key**

To read an object by its key, you use the get() method of the object store. The following getContactById() function finds a contact by an id:

function getContactById(db, id) {

const txn = db.transaction('Contacts', 'readonly');

const store = txn.objectStore('Contacts');

let query = store.get(id);

query.onsuccess = (event) => {

if (!event.target.result) {

console.log(`The contact with ${id} not found`);

} else {

console.table(event.target.result);

}

};

query.onerror = (event) => {

console.log(event.target.errorCode);

}

txn.oncomplete = function () {

db.close();

};

};

Code language: JavaScript (javascript)

When you call the get() method of the object store, it returns a query that will execute asynchronously.

Because the query can succeed or fail, you need to assign the onsuccess and onerror handlers to handle each case.

If the query succeeded, you’ll get the result in the event.target.result. Othewise, you’ll get error code via event.target.errorCode.

The following code closes the connection to the database once the transaction completes:

txn.oncomplete = function () {

db.close();

};

Code language: JavaScript (javascript)

Actually, the database connection is closed only when all the transactions created using it completed.

The following calls the getContactById() in the onsuccess event handler to get the contact with id 1:

request.onsuccess = (event) => {

const db = event.target.result;

getContactById(db, 1);

};

Code language: JavaScript (javascript)

Output:

Table

Description automatically generated

### **6) Read data from the object store by an index**

The following defines a new function called getContactByEmail() that uses the email index to query data:

function getContactByEmail(db, email) {

const txn = db.transaction('Contacts', 'readonly');

const store = txn.objectStore('Contacts');

*// get the index from the Object Store*

const index = store.index('email');

*// query by indexes*

let query = index.get(email);

*// return the result object on success*

query.onsuccess = (event) => {

console.log(query.result); *// result objects*

};

query.onerror = (event) => {

console.log(event.target.errorCode);

}

*// close the database connection*

txn.oncomplete = function () {

db.close();

};

}

Code language: JavaScript (javascript)

How it works.

* First, get the email index object from the Contacts object store.
* Second, use the index to read the data by calling the get() method.
* Third, show the result the console in the onsuccess event handler of the query.

The following illustrates how to use the getContactByEmail() function in the onsuccess event handler:

request.onsuccess = (event) => {

const db = event.target.result;

*// get contact by email*

getContactByEmail(db, 'jane.doe@gmail.com');

};

Code language: JavaScript (javascript)

Output:

Graphical user interface, application

Description automatically generated with medium confidence

### **7) Read all data from an object store**

The following shows how to use a cursor to read all the objects from the Contacts object store:

function getAllContacts(db) {

const txn = db.transaction('Contacts', "readonly");

const objectStore = txn.objectStore('Contacts');

objectStore.openCursor().onsuccess = (event) => {

let cursor = event.target.result;

if (cursor) {

let contact = cursor.value;

console.log(contact);

*// continue next record*

cursor.continue();

}

};

*// close the database connection*

txn.oncomplete = function () {

db.close();

};

}

Code language: JavaScript (javascript)

The objectStore.openCursor() returns a cursor used to iterate over an object store.

To iterate over the objects in an object store using the cursor, you need to assign an onsuccess handler:

objectStore.openCursor().onsuccess = (event) => {

*//...*

};

Code language: JavaScript (javascript)

The event.target.result returns the cursor. To get the data, you use the cursor.value property.

The cursor.continue() method advances the cursor to the position of the next record in the object store.

The following calls the getAllContacts() in the onsuccess event handler to show all data from the Contacts object store:

request.onsuccess = (event) => {

const db = event.target.result;

*// get all contacts*

getAllContacts(db);

};

Code language: JavaScript (javascript)

Output:

Text

Description automatically generated

### **8) Delete a contact**

To delete a record from the object store, you use the delete() method of the object store.

The following function deletes a contact by its id from the Contacts object store:

function deleteContact(db, id) {

*// create a new transaction*

const txn = db.transaction('Contacts', 'readwrite');

*// get the Contacts object store*

const store = txn.objectStore('Contacts');

*//*

let query = store.delete(id);

*// handle the success case*

query.onsuccess = function (event) {

console.log(event);

};

*// handle the error case*

query.onerror = function (event) {

console.log(event.target.errorCode);

}

*// close the database once the*

*// transaction completes*

txn.oncomplete = function () {

db.close();

};

}

Code language: JavaScript (javascript)

And you can call the deleteContact() function in the onsuccess event handler to delete the contact with id 1 as follows:

request.onsuccess = (event) => {

const db = event.target.result;

deleteContact(db, 1);

};

Code language: JavaScript (javascript)

If you run the code, you’ll find that the contact with the id 1 will be deleted.

### **Put it all together**

The following shows the complete app.js file:

(function () {

*// check for IndexedDB support*

if (!window.indexedDB) {

console.log(`Your browser doesn't support IndexedDB`);

return;

}

*// open the CRM database with the version 1*

const request = indexedDB.open('CRM', 1);

*// create the Contacts object store and indexes*

request.onupgradeneeded = (event) => {

let db = event.target.result;

*// create the Contacts object store*

*// with auto-increment id*

let store = db.createObjectStore('Contacts', {

autoIncrement: true

});

*// create an index on the email property*

let index = store.createIndex('email', 'email', {

unique: true

});

};

*// handle the error event*

request.onerror = (event) => {

console.error(`Database error: ${event.target.errorCode}`);

};

*// handle the success event*

request.onsuccess = (event) => {

const db = event.target.result;

*// insert contacts*

*// insertContact(db, {*

*// email: 'john.doe@outlook.com',*

*// firstName: 'John',*

*// lastName: 'Doe'*

*// });*

*// insertContact(db, {*

*// email: 'jane.doe@gmail.com',*

*// firstName: 'Jane',*

*// lastName: 'Doe'*

*// });*

*// get contact by id 1*

*// getContactById(db, 1);*

*// get contact by email*

*// getContactByEmail(db, 'jane.doe@gmail.com');*

*// get all contacts*

*// getAllContacts(db);*

deleteContact(db, 1);

};

function insertContact(db, contact) {

*// create a new transaction*

const txn = db.transaction('Contacts', 'readwrite');

*// get the Contacts object store*

const store = txn.objectStore('Contacts');

*//*

let query = store.put(contact);

*// handle success case*

query.onsuccess = function (event) {

console.log(event);

};

*// handle the error case*

query.onerror = function (event) {

console.log(event.target.errorCode);

}

*// close the database once the*

*// transaction completes*

txn.oncomplete = function () {

db.close();

};

}

function getContactById(db, id) {

const txn = db.transaction('Contacts', 'readonly');

const store = txn.objectStore('Contacts');

let query = store.get(id);

query.onsuccess = (event) => {

if (!event.target.result) {

console.log(`The contact with ${id} not found`);

} else {

console.table(event.target.result);

}

};

query.onerror = (event) => {

console.log(event.target.errorCode);

}

txn.oncomplete = function () {

db.close();

};

};

function getContactByEmail(db, email) {

const txn = db.transaction('Contacts', 'readonly');

const store = txn.objectStore('Contacts');

*// get the index from the Object Store*

const index = store.index('email');

*// query by indexes*

let query = index.get(email);

*// return the result object on success*

query.onsuccess = (event) => {

console.table(query.result); *// result objects*

};

query.onerror = (event) => {

console.log(event.target.errorCode);

}

*// close the database connection*

txn.oncomplete = function () {

db.close();

};

}

function getAllContacts(db) {

const txn = db.transaction('Contacts', "readonly");

const objectStore = txn.objectStore('Contacts');

objectStore.openCursor().onsuccess = (event) => {

let cursor = event.target.result;

if (cursor) {

let contact = cursor.value;

console.log(contact);

*// continue next record*

cursor.continue();

}

};

*// close the database connection*

txn.oncomplete = function () {

db.close();

};

}

function deleteContact(db, id) {

*// create a new transaction*

const txn = db.transaction('Contacts', 'readwrite');

*// get the Contacts object store*

const store = txn.objectStore('Contacts');

*//*

let query = store.delete(id);

*// handle the success case*

query.onsuccess = function (event) {

console.log(event);

};

*// handle the error case*

query.onerror = function (event) {

console.log(event.target.errorCode);

}

*// close the database once the*

*// transaction completes*

txn.oncomplete = function () {

db.close();

};

}

})();

Code language: JavaScript (javascript)

## **Summary**

* The IndexedDB is a large scale object store built in the web browsers.
* The IndexedDB stores data as key-value pairs. The values can be any data including simple and complex ones.
* The IndexedDB consists of one or more databases. Each database has one or more object stores. Typically, you create a database in the IndexedDB per web application.
* The IndexedDB is useful for web applications that don’t require a persistent internet connection, especially for applications that work both online and offline.

# JavaScript FormData

**Summary**: in this tutorial, you’ll learn about the JavaScript FormData API and how to post the FormData using fetch API.

## **Introduction to the JavaScript FormData API**

Suppose you have a subscription form with two fields name and email.

<form action="subscribe.php" method="post" id="subscription">

**<div>**

**<label for="name">**Name:**</label>**

**<input type="text" name="name" id="name"/>**

**</div>**

**<div>**

**<label for="email">**Email:**</label>**

**<input type="email" name="email" id="email"/>**

**</div>**

**<div>**

**<button type="submit" id="submit">**Subscribe**</button>**

**</div>**

</form>

Code language: JavaScript (javascript)

When you click the submit button, the web browser submits the values of the name and email fields in the form to the server.

Similarly, the FormData interface lets you construct a set of key/value pairs representing form fields and their values in JavaScript.

Once having a FormData object, you can post it to the server using the [fetch API](https://www.javascripttutorial.net/javascript-fetch-api/). If you want to submit the form as if it were like the GET request, you can pass the FormData object to the URLSearchParams constructor.

## **Create a FormData object**

The following creates a new FormData object from an HTML form element:

const formData = new FormData(form);

Code language: JavaScript (javascript)

The following script shows the values of a FormData object:

const btn = document.querySelector('#submit');

const form = document.querySelector('#subscription');

btn.addEventListener('click', (e) => {

*// prevent the form from submitting*

e.preventDefault();

*// show the form values*

const formData = new FormData(form);

const values = [...formData.entries()];

console.log(values);

});

Code language: JavaScript (javascript)

How it works.

* First, select the submit button using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method of the document object.
* Next, add an event handler to handle the click event of the submit button.
* Then, call the [preventDefault()](https://www.javascripttutorial.net/dom/events/prevent-default-action-of-events/) method of the event object to avoid form submission.
* After that, create a new FormData object from the form element.
* Finally, call the entries() method of the FormData object. Since the entries() method returns an iterator, you must use the [spread operator](https://www.javascripttutorial.net/es6/javascript-spread/) (...).

## **FormData methods**

The FormData object has the following methods:

### **append() method**

The append() method has two forms:

FormData.append(name, value)

FormData.append(name, value, file)

Code language: JavaScript (javascript)

The append() method appends a new value onto an existing key inside a FormData object or adds the key if it does not exist.

### **delete() method**

FormData.delete(name);

Code language: JavaScript (javascript)

The delete() method deletes a key/value pair by the name of the key from the FormData object.

### **entries() method**

FormData.entries()

Code language: JavaScript (javascript)

The entries() method returns an [iterator](https://www.javascripttutorial.net/es6/javascript-iterator/) of all key/value pairs in the FormData object.

### **get() method**

FormData.get(name)

Code language: JavaScript (javascript)

The get() method returns the first value by a key name.

### **getAll() method**

FormData.getAll(name)

Code language: JavaScript (javascript)

The getAll() method returns an array of all the values by a key name.

### **has() method**

FormData.has(name);

Code language: JavaScript (javascript)

The has() method returns true if the FormData object contains a key name.

### **keys() method**

FormData.keys()

Code language: JavaScript (javascript)

The keys() method returns an iterator of all the keys.

### **set() method**

The set() method has two forms:

FormData.set(name, value);

FormData.set(name, value, filename);

Code language: JavaScript (javascript)

The set() method sets a new value for an existing key name or adds the key/value if it does not already exist.

### **values() method**

FormData.values()

Code language: JavaScript (javascript)

The values() method returns an iterator of all the values in the FormData object.

### **Submit the FormData using fetch API**

We’ll build a simple [subscription form that uses the FetchAPI to post a FormData object](https://www.javascripttutorial.net/sample/webapis/formdata/) to the server.

The following illustrates how to submit the FormData using the fetch API:

const btn = document.querySelector('#submit');

const form = document.querySelector('#subscription');

const messageEl = document.querySelector('#message');

btn.addEventListener('click', (e) => {

e.preventDefault();

subscribe();

});

const subscribe = async () => {

try {

let response = await fetch('subscribe.php', {

method: 'POST',

body: new FormData(form),

});

const result = await response.json();

showMessage(result.message, response.status == 200 ? 'success' : 'error');

} catch (error) {

showMessage(error.message, 'error');

}

};

const showMessage = (message, type = 'success') => {

messageEl.innerHTML = `

<div class="alert alert-${type}">

${message}

</div>

`;

};

Code language: JavaScript (javascript)

In this example, we define a function named subscribe() and call it in the submit button’s click event listener. The subscribe() function is an [async function](https://www.javascripttutorial.net/es-next/javascript-async-await/) because it uses the await keyword.

Inside the subscribe() function:

First, post the form data to the subscribe.php script using the fetch() method:

let response = await fetch('subscribe.php', {

method: 'POST',

body: new FormData(form),

});

Code language: JavaScript (javascript)

Second, get the JSON response by calling the json() method of the response object:

const result = await response.json();

Code language: JavaScript (javascript)

Third, show a success message if the HTTP status code is 200. Otherwise, display an error message. The message property of the result stores the success or error message.

showMessage(result.message, response.status == 200 ? 'success' : 'error');

Code language: JavaScript (javascript)

Finally, define the showMessage() function that displays a success or error message:

const showMessage = (message, type = 'success') => {

messageEl.innerHTML = `

<div class="alert alert-${type}">

${message}

</div>

`;

};

Code language: JavaScript (javascript)

## **Put it all together.**

index.html

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8" />**

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

**<link rel="stylesheet" href="css/style.css" />**

**<title>**JavaScript FormData Demo**</title>**

**</head>**

**<body>**

**<main>**

**<div id="message"></div>**

**<form action="subscribe.php" method="post" id="subscription">**

**<div>**

**<label for="name">**Name:**</label>**

**<input type="text" name="name" id="name"/>**

**</div>**

**<div>**

**<label for="email">**Email:**</label>**

**<input type="email" name="email" id="email"/>**

**</div>**

**<div>**

**<button type="submit" id="submit">**Subscribe**</button>**

**</div>**

**</form>**

**</main>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: JavaScript (javascript)

app.js

const btn = document.querySelector('#submit');

const form = document.querySelector('#subscription');

const messageEl = document.querySelector('#message');

btn.addEventListener('click', (e) => {

e.preventDefault();

subscribe();

});

const subscribe = async () => {

try {

let response = await fetch('subscribe.php', {

method: 'POST',

body: new FormData(form),

});

const result = await response.json();

showMessage(result.message, response.status == 200 ? 'success' : 'error');

} catch (error) {

showMessage(error.message, 'error');

}

};

const showMessage = (message, type = 'success') => {

messageEl.innerHTML = `

<div class="alert alert-${type}">

${message}

</div>

`;

};

Code language: JavaScript (javascript)

## **Summary**

* Use the JavaScript FormData API to capture the HTML form values.
* Use the Fetch API to submit the FormData to the server.

# JavaScript Drag and Drop

**Summary**: in this tutorial, you’ll learn about the JavaScript drag and drop API and how to use it to implement a simple drag&drop application.

## **Introduction to JavaScript Drag and Drop API**

HTML5 formally introduced the drag-and-drop specification. Most modern web browsers have implemented native drag-and-drop based on the HTML5 spec.

By default, only image and text can be draggable. To drag an image, you simply hold the mouse button down and then move it. To drag the text, you need to highlight some text and drag it in the same way as you would drag an image.

HTML5 spec specifies that almost all elements can be draggable. To make an element draggable, you add the draggable property with the value of true to its HTML tag. For example:

**<div class="item" draggable="true"></div>**

Code language: HTML, XML (xml)

## **Events on draggable elements**

When you drag an element, these events fire in the following sequence:

* dragstart
* drag
* dragend

When you hold a mouse button and begin to move the mouse, the dragstart event fires on the draggable element that you’re dragging. The cursor changes to a no-drop symbol (a circle with a line through it) to indicate that you cannot drop the element on itself.

After the dragstart event fires, the drag event fires repeatedly as long as you drag the element.

And The dragend event fires when you stop dragging the element.

The target of all events (e.target) is the element that is being dragged.

By default, the browser doesn’t change the appearance of the dragged element. Therefore, you can customize its appearance based on your preferences.

## **Events on drop targets**

When you drag an element over a valid drop target, these events fire in the following sequence:

* dragenter
* dragover
* dragleave or drop

The dragenter event fires as soon as you drag the element over a drop target.

After the dragenter event fires, the dragover event fires repeatedly as long as you’re dragging the element within the boundary of the drop target.

When you drag the element outside of the boundary of the drop target, the dragover event stops firing and the dragleave event fires.

In case you drop the element on the target, the drop event fires instead of the dragleave event.

The target (e.target) of the dragenter, dragover, dragleave, and drop events are the drop target elements.

## **Valid drop target**

Almost all elements support the drop target events (dragenter, dragover, dragleave, and drop). However, they don’t allow dropping by default.

If you drop an element over a drop target that doesn’t allow to drop, the drop event won’t fire.

To turn an element into a valid drop target, you can override the default behavior of both dragenter and dragover events by calling the event.preventDefault() method in their corresponding event handlers. (See the example section for more information)

## **Transfer data using dataTransfer object**

To transfer data in a drag-and-drop action, you use the dataTransfer object.

The dataTransfer object is a property of the event. It allows you to transfer data from the dragged element to the drop target.

The dataTransfer object has two methods: setData() and getData().

The setData() allows you to set the drag operation’s data to the specified format and data:

dataTransfer.setData(format, data)

Code language: CSS (css)

The format can be text/plain or text/uri-list. And the data can be a string representing the data to add to the drag object.

The getData() method retrieves the drag data stored by the setData() method.

The getData() accepts one argument:

dataTransfer(format)

The format can be text/plain or text/uri-list. The getData() returns a string stored by the setData() method or an empty string if the drag operation doesn’t include data.

## **JavaScript drag and drop example**

We’ll develop the following [simple drag-and-drop application](https://www.javascripttutorial.net/sample/webapis/drag-n-drop-basics/) to demonstrate the JavaScript drag and drop API:

### **Create the project structure**

First, create a new folder called drag-n-drop-basics. Inside this folder, create two subfolders called css and js.

Second, create a new file called app.js in the js folder, style.css in the css folder, and index.html in the drag-n-drop-basics folder.

Third, place the link to the style.css and script tag that links to the app.js in the index.html file like this:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript - Drag and Drop Demo**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

For the CSS, you can [grab it from here](https://www.javascripttutorial.net/sample/webapis/drag-n-drop-basics/css/style.css).

### **Build the index.html file**

Place the following code in the index.html file:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript - Drag and Drop Demo**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<div class="container">**

**<h1>**JavaScript - Drag and Drop**</h1>**

**<div class="drop-targets">**

**<div class="box">**

**<div class="item" id="item">**

**</div>**

**</div>**

**<div class="box"></div>**

**<div class="box"></div>**

**</div>**

**</div>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this index.html file, we used the .container element to align the heading and drop-targets element.

Inside the drop-targets element, we placed three div elements with the same class box. And we place another div element with the class item in the first box.

If you open the index.html and try to drag the yellow box, you’ll see the cursor indicating that you cannot drag:

A picture containing graphical user interface

Description automatically generated

To make the element draggable, you add the draggable property with the value of true to its HTML tag as follows:

<div class="item" id="item" draggable="true">

Code language: JavaScript (javascript)

Now, if you save the index.html, open it in the browser again, you’ll see that you can drag the item element like this:

A picture containing diagram

Description automatically generated

### **Handle events on the draggable element**

The style.css file has the .hide class that hides an element:

.hide {

display: none;

}

Code language: CSS (css)

In the app.js file, you add the following code:

*// select the item element*

const item = document.querySelector('.item');

*// attach the dragstart event handler*

item.addEventListener('dragstart', dragStart);

*// handle the dragstart*

function dragStart(e) {

console.log('drag starts...');

}

Code language: JavaScript (javascript)

How it works:

* First, select the draggable element using the querySelector().
* Second, attach a dragstart event handler to the draggable element.
* Third, define the dragStart() function to handle the dragstart event.

If you open the index.html file and start dragging the draggable element, you’ll see the drag starts... message in the console.

In the dragStart event handler, you need to store the id of the draggable element. And you need to hide it:

function dragStart(e) {

e.dataTransfer.setData('text/plain', e.target.id);

e.target.classList.add('hide');

}

Code language: JavaScript (javascript)

If you drag the element, you’ll see that it disappears once you start dragging.

To solve this, you use the [setTimeout()](https://www.javascripttutorial.net/javascript-bom/javascript-settimeout/) function:

function dragStart(e) {

e.dataTransfer.setData('text/plain', e.target.id);

setTimeout(() => {

e.target.classList.add('hide');

}, 0);

}

Code language: JavaScript (javascript)

Now, you can drag the draggable element out of its original position:

Graphical user interface, application

Description automatically generated

### **Handle events on drop targets**

The style.css file also has a CSS class called .drag-over that turns the border style of the drop target into dashed and red:

.drag-over {

border: dashed 3px red;

}

Code language: CSS (css)

In the app.js, you need to select the drop target elements and handle the dragenter, dragover, dragleave, and drop events of these elements:

const boxes = document.querySelectorAll('.box');

boxes.forEach(box => {

box.addEventListener('dragenter', dragEnter)

box.addEventListener('dragover', dragOver);

box.addEventListener('dragleave', dragLeave);

box.addEventListener('drop', drop);

});

function dragEnter(e) {

}

function dragOver(e) {

}

function dragLeave(e) {

}

function drop(e) {

}

Code language: JavaScript (javascript)

The border style of the drop target should change when the dragenter and dragover event occurs. It should restore the style when the dragleave and drop event occurs.

To do it, you add and remove the drag-over class to the drop target like this:

function dragEnter(e) {

e.target.classList.add('drag-over');

}

function dragOver(e) {

e.target.classList.add('drag-over');

}

function dragLeave(e) {

e.target.classList.remove('drag-over');

}

function drop(e) {

e.target.classList.remove('drag-over');

}

Code language: JavaScript (javascript)

Now, if you drag the draggable element to another drop target, you’ll see that the border of the drop target changes as shown in the following picture:

A picture containing rectangle

Description automatically generated

To make the drop target valid, you need to call event.preventDefault() in the dragenter and dragover event handlers like this:

function dragEnter(e) {

e.preventDefault();

e.target.classList.add('drag-over');

}

function dragOver(e) {

e.preventDefault();

e.target.classList.add('drag-over');

}

Code language: JavaScript (javascript)

If you don’t do this, the drop event will never fire because the div element is not a valid drop target by default.

If you drag the draggable element to a drop target, you’ll see that the cursor changes indicating that you can drop the element:

Chart

Description automatically generated with medium confidence

Now, if you drop the item element, you’ll see that it disappears immediately.

To solve this issue, you need to add handle the drop event.

* First, get the id of the draggable element using the getData() method of the dataTransfer object.
* Second, append the draggable element as a child element of the drop target element.
* Third, remove the hide class from the draggable element.

The following code shows the complete drop event handler:

function drop(e) {

e.target.classList.remove('drag-over');

*// get the draggable element*

const id = e.dataTransfer.getData('text/plain');

const draggable = document.getElementById(id);

*// add it to the drop target*

e.target.appendChild(draggable);

*// display the draggable element*

draggable.classList.remove('hide');

}

Code language: JavaScript (javascript)

If you drag and drop the draggable element now, it should work as expected.

The following shows the complete app.js file:

*/\* draggable element \*/*

const item = document.querySelector('.item');

item.addEventListener('dragstart', dragStart);

function dragStart(e) {

e.dataTransfer.setData('text/plain', e.target.id);

setTimeout(() => {

e.target.classList.add('hide');

}, 0);

}

*/\* drop targets \*/*

const boxes = document.querySelectorAll('.box');

boxes.forEach(box => {

box.addEventListener('dragenter', dragEnter)

box.addEventListener('dragover', dragOver);

box.addEventListener('dragleave', dragLeave);

box.addEventListener('drop', drop);

});

function dragEnter(e) {

e.preventDefault();

e.target.classList.add('drag-over');

}

function dragOver(e) {

e.preventDefault();

e.target.classList.add('drag-over');

}

function dragLeave(e) {

e.target.classList.remove('drag-over');

}

function drop(e) {

e.target.classList.remove('drag-over');

*// get the draggable element*

const id = e.dataTransfer.getData('text/plain');

const draggable = document.getElementById(id);

*// add it to the drop target*

e.target.appendChild(draggable);

*// display the draggable element*

draggable.classList.remove('hide');

}

Code language: JavaScript (javascript)

And [here is the link to the demo](https://www.javascripttutorial.net/sample/webapis/drag-n-drop-basics/).

## **Summary**

* Add the draggable property with the value of true to an element to make it draggable.
* The dragstart, drag, and dragend events fire on the draggable element.
* The dragenter, dragover, dragleave or drop events fire on the drop target.
* Call the event.preventDefault() on the dragenter and dragover event handlers to make an element a valid drop target.
* Use the event.dataTransfer object with the setData() and getData() methods to transfer data in the drag-and-drop operation.

# JavaScript FileReader

**Summary**: in this tutorial, you’ll learn about the JavaScript FileReader API and how to use it to implement the file upload.

## **Introduction to the JavaScript FileReader API**

When you [drag and drop](https://www.javascripttutorial.net/web-apis/javascript-drag-and-drop/) files to the web browser or select files to upload via the file input element, JavaScript represents each file as a File object.

The File object allows you to access the selected file in JavaScript. And JavaScript uses the FileList object to hold the File objects.

To read the content of a file, you use the FileReader object. Note that the FileReader only can access the files you selected via drag & drop or file input.

To use the FileReader object, you follow these steps:

First, create a new FileObject:

const reader = new FileReader();

Code language: JavaScript (javascript)

Second, call one of the read methods to read the content of a file. For example:

reader.readAsDataURL(file);

Code language: JavaScript (javascript)

The readAsDataURL() method reads the content of the file, which you get from the FileList object.

The readAsDataURL() method returns an object with the result property that contains the data as a data: URL. The data:URL represents the file’s data as a base64 encoded string.

For example, you can use the readAsDataURL() to read an image and show its base64 encoded string on a webpage.

Besides the readAsDataURL() method, the FileReader has other methods for reading file’s data such as readAsText(), readAsBinaryString(), and readAsArrayBuffer().

Since all of these methods read the file’s data asynchronously, you cannot just return the result like this:

const data = reader.readAsDataURL(file);

Code language: JavaScript (javascript)

When the readAsDataURL() method completes reading the file successfully, the FileReader fires the load event.

Third, add an event handler to handle the load event of the FileReader object:

reader.addEventListener('load', (e) => {

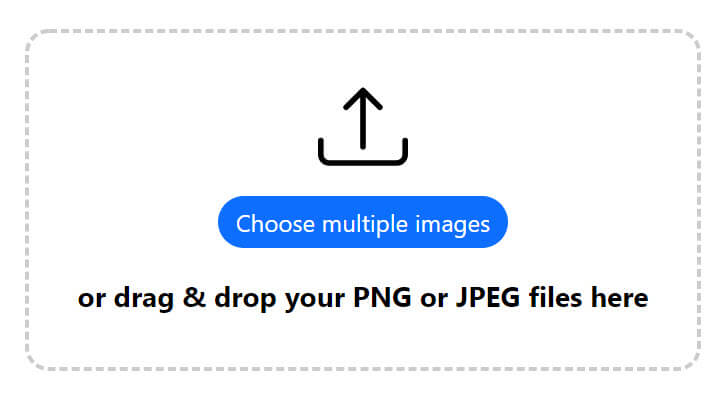
const data = e.target.result;

}

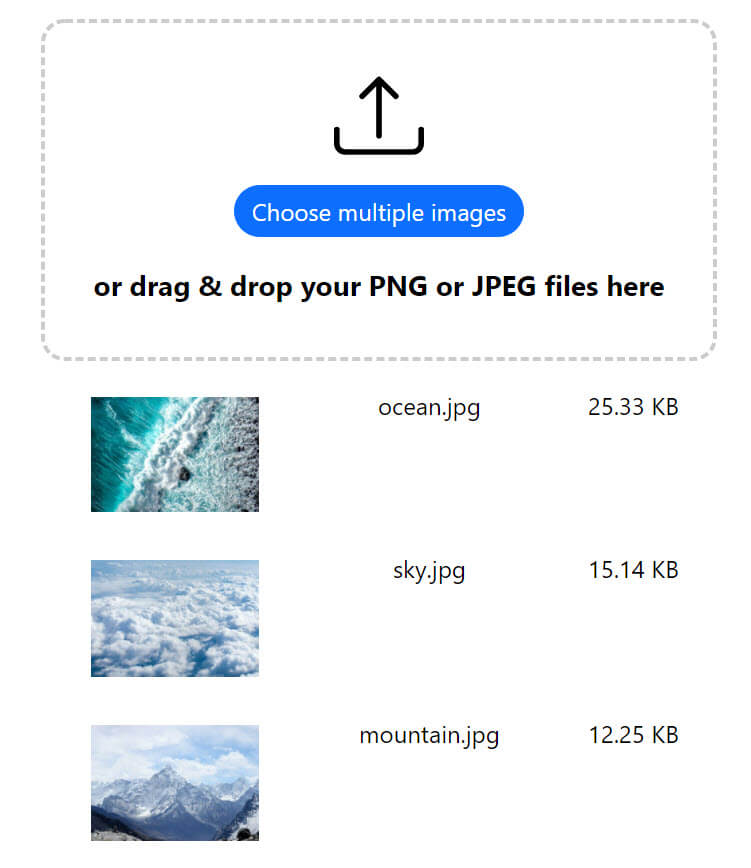
Code language: JavaScript (javascript)

## **Using JavaScript FileReader to implement an image upload application**

We’ll use the FileReader to implement an [image upload application](https://www.javascripttutorial.net/sample/webapis/filereader/):

[](https://www.javascripttutorial.net/sample/webapis/filereader/)

When you drag & drop images to the drop zone, the application will use the FileReader to read the images and show them on the page together with the filename and filesize:

[](https://www.javascripttutorial.net/sample/webapis/filereader/)

Also, the application will use the [Fetch API](https://www.javascripttutorial.net/javascript-fetch-api/) to upload the files to the server.

For the server-side, we’ll implement a simple PHP script that uploads the images to the 'uploads' folder on the server.

### **Setting up the project structure**

First, create the following file and directory structure:

├── css

| └── style.css

├── images

| └── upload.svg

├── js

| └── app.js

├── index.html

├── upload.php

└── uploads

Code language: JavaScript (javascript)

### **index.html**

The following shows the index.html file:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8" />**

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

**<link rel="stylesheet" href="css/style.css" />**

**<title>**FileReader API Demo - Image Upload Application**</title>**

**</head>**

**<body>**

**<main>**

**<div class="dropzone">**

**<img src="images/upload.svg" alt="upload" width="60" />**

**<input type="file" class="files" id="images"**

**accept="image/png, image/jpeg"**

**multiple />**

**<label for="images">**Choose multiple images**</label>**

**<h3>**or drag & drop your PNG or JPEG files here**</h3>**

**</div>**

**<div class="image-list"></div>**

**</main>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: JavaScript (javascript)

In the index.html, we add the css/style.css to the head of the html document and js/app.js before the enclosing body tag.

The div element with the class dropzone allows you to drag and drop images into it. Also, the file input element will enable you to select files to upload.

The file input element accepts multiple files and only allows jpeg and png images:

<input type="file" class="files" id="images"

accept="image/png, image/jpeg"

multiple />

Code language: JavaScript (javascript)

The style.css provides the styles that transform the file input element into a button. Also, it has the active class that highlights the dropzone when you drag the file into it.

The div element with the image-list class will show the uploaded images.

### **app.js**

First, select dropzone, file input (files), and image list elements using the querySelector() method:

const imageList = document.querySelector('.image-list');

const fileInput = document.querySelector('.files');

const dropzone = document.querySelector('.dropzone');

Code language: JavaScript (javascript)

Second, define a function that adds the active class to or remove it from the dropzone:

const setActive = (dropzone, active = true) => {

const hasActiveClass = dropzone.classList.contains('active');

if (active && !hasActiveClass) {

return dropzone.classList.add('active');

}

if (!active && hasActiveClass) {

return dropzone.classList.remove('active');

}

};

Code language: JavaScript (javascript)

If you call setActive(dropzone), it’ll add the active class to the dropzone. If you call setActive(dropzone, false), it’ll remove the active class from the dropzone.

Third, highlight the dropzone when the dragenter and dragover events occur and remove the highlight when the dragleave and drop events occur:

dropzone.addEventListener('dragenter', (e) => {

e.preventDefault();

setActive(dropzone);

});

dropzone.addEventListener('dragover', (e) => {

e.preventDefault();

setActive(dropzone);

});

dropzone.addEventListener('dragleave', (e) => {

e.preventDefault();

setActive(dropzone, false);

});

dropzone.addEventListener('drop', (e) => {

e.preventDefault();

setActive(dropzone, false);

*// ..*

});

Code language: JavaScript (javascript)

Fourth, get the FileList object in the e.target as e.target.files in the drop event handler of dropzone:

dropzone.addEventListener('drop', (e) => {

e.preventDefault();

setActive(dropzone, false);

*// get the FileList*

const { files } = e.dataTransfer;

handleImages(files);

});

Code language: JavaScript (javascript)

In the drop event handler, we use object destructuring to get the FileList object and call the handleImages() function to handle the uploaded images:

Fifth, define the handleImages() function:

const handleImages = (files) => {

*// get valid images*

let validImages = [...files].filter((file) =>

['image/jpeg', 'image/png'].includes(file.type)

);

*// show the image*

validImages.forEach(showImage);

*// upload all images*

uploadImages(validImages);

};

Code language: JavaScript (javascript)

The handleImages() function gets the valid images, shows each valid image on the page using the showImage() function and uploads all the images to the server using the uploadImages() function.

Sixth, define the showImage() function that shows each image in the validImages array:

const showImage = (image) => {

const reader = new FileReader();

reader.readAsDataURL(image);

reader.addEventListener('load', (e) => {

const div = document.createElement('div');

div.classList.add('image');

div.innerHTML = `

<img src="${e.target.result}" alt="${image.name}">

<p>${image.name}</p>

<p>${formatBytes(image.size)}</p>

`;

imageList.appendChild(div);

});

};

Code language: JavaScript (javascript)

The showImage() uses the FileReader to read the uploaded image as the data URL. Once the FileReader completes reading the file, it’ll create a new div element to hold the image information.

Note that the formatBytes() function converts the size in bytes into a human-readable format:

function formatBytes(size, decimals = 2) {

if (size === 0) return '0 bytes';

const k = 1024;

const dm = decimals < 0 ? 0 : decimals;

const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'];

const i = Math.floor(Math.log(size) / Math.log(k));

return parseFloat((size / Math.pow(k, i)).toFixed(dm)) + ' ' + sizes[i];

}

Code language: JavaScript (javascript)

Seventh, define the uploadImages() function that uploads all images to the server:

const uploadImages = async (images) => {

const formData = new FormData();

[...images].forEach((image) =>

formData.append('images[]', image, image.name)

);

const response = await fetch('upload.php', {

method: 'POST',

body: formData,

});

return await response.json();

};

Code language: JavaScript (javascript)

The uploadImages() function uses the FormData API to construct data for submission:

const formData = new FormData();

Code language: JavaScript (javascript)

For each image, we add it to the FormData object:

[...images].forEach((image) =>

formData.append('images[]', image, image.name)

);

Code language: JavaScript (javascript)

Note that the images variable is a FileList object, not an array. To use the forEach() method, you use the spread operator (...) to convert the FileList object to an array like this:

[...images]

Code language: JavaScript (javascript)

All key/value pairs in the form data have the same key as images[]; in PHP, you can access it as an array ($\_FILES['images'])

The uploadImages() function uses the Fetch API to upload the images (as the FormData object) to the server:

const response = await fetch('upload.php', {

method: 'POST',

body: formData,

});

return await response.json();

Code language: JavaScript (javascript)

Eighth, add the change event handler to the file input element if users select files using this input element:

fileInput.addEventListener('change', (e) => {

const { files } = e.target;

handleImages(files);

});

Code language: JavaScript (javascript)

In the change event handler, you can access the FileList object as e.target.files. The logic for showing and uploading images is the same as drag & drop.

Note that if you drag & drop images outside the dropzone, the web browser will display the images by default.

To prevent this, you call the preventDefault() method of the dragover and drop event objects of the document like this:

*// prevent the drag & drop on the page*

document.addEventListener('dragover', (e) => e.preventDefault());

document.addEventListener('drop', (e) => e.preventDefault());

Code language: JavaScript (javascript)

The following shows the complete app.js file:

const imageList = document.querySelector('.image-list');

const fileInput = document.querySelector('.files');

const dropzone = document.querySelector('.dropzone');

const setActive = (dropzone, active = true) => {

*// active class*

const hasActiveClass = dropzone.classList.contains('active');

if (active && !hasActiveClass) {

return dropzone.classList.add('active');

}

if (!active && hasActiveClass) {

return dropzone.classList.remove('active');

}

};

dropzone.addEventListener('dragenter', (e) => {

e.preventDefault();

setActive(dropzone);

});

dropzone.addEventListener('dragover', (e) => {

e.preventDefault();

setActive(dropzone);

});

dropzone.addEventListener('dragleave', (e) => {

e.preventDefault();

setActive(dropzone, false);

});

dropzone.addEventListener('drop', (e) => {

e.preventDefault();

setActive(dropzone, false);

*// get the valid files*

const { files } = e.dataTransfer;

*// hand images*

handleImages(files);

});

const handleImages = (files) => {

*// get valid images*

let validImages = [...files].filter((file) =>

['image/jpeg', 'image/png'].includes(file.type)

);

*// show the image*

validImages.forEach(showImage);

*// upload files*

uploadImages(validImages);

};

const showImage = (image) => {

const reader = new FileReader();

reader.readAsDataURL(image);

reader.addEventListener('load', (e) => {

const div = document.createElement('div');

div.classList.add('image');

div.innerHTML = `

<img src="${e.target.result}" alt="${image.name}">

<p>${image.name}</p>

<p>${formatBytes(image.size)}</p>

`;

imageList.appendChild(div);

});

};

const uploadImages = async (images) => {

const formData = new FormData();

[...images].forEach((image) =>

formData.append('images[]', image, image.name)

);

const response = await fetch('upload.php', {

method: 'POST',

body: formData,

});

return await response.json();

};

function formatBytes(size, decimals = 2) {

if (size === 0) return '0 bytes';

const k = 1024;

const dm = decimals < 0 ? 0 : decimals;

const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'];

const i = Math.floor(Math.log(size) / Math.log(k));

return parseFloat((size / Math.pow(k, i)).toFixed(dm)) + ' ' + sizes[i];

}

fileInput.addEventListener('change', (e) => {

const { files } = e.target;

handleImages(files);

});

*// prevent the drag & drop on the page*

document.addEventListener('dragover', (e) => e.preventDefault());

document.addEventListener('drop', (e) => e.preventDefault());

Code language: JavaScript (javascript)

Finally, create a simple upload.php script that moves the uploaded images to the uploads folder:

<?php

const APP\_ROOT = 'http://localhost:8080/';

const UPLOAD\_DIR = \_\_DIR\_\_ . '/uploads';

const MESSAGES = [

UPLOAD\_ERR\_OK => 'File uploaded successfully',

UPLOAD\_ERR\_INI\_SIZE => 'File is too big to upload',

UPLOAD\_ERR\_FORM\_SIZE => 'File is too big to upload',

UPLOAD\_ERR\_PARTIAL => 'File was only partially uploaded',

UPLOAD\_ERR\_NO\_FILE => 'No file was uploaded',

UPLOAD\_ERR\_NO\_TMP\_DIR => 'Missing a temporary folder on the server',

UPLOAD\_ERR\_CANT\_WRITE => 'File is failed to save to disk.',

UPLOAD\_ERR\_EXTENSION => 'File is not allowed to upload to this server',

];

const ALLOWED\_FILES = [

'image/png' => 'png',

'image/jpeg' => 'jpg'

];

const MAX\_SIZE = 5 \* 1024 \* 1024; *// 5MB*

const HTTP\_STATUSES = [

200 => 'OK',

400 => 'Bad Request',

404 => 'Not Found',

405 => 'Method Not Allowed'

];

$is\_post\_request = strtolower($\_SERVER['REQUEST\_METHOD']) === 'post';

$has\_files = isset($\_FILES['images']);

if (!$is\_post\_request || !$has\_files) {

response(405, [

'success' => false,

'message' => ' Method not allowed or files do not exist'

]);

}

$files = $\_FILES['images'];

$file\_count = count($files['name']);

*// validation*

$errors = [];

for ($i = 0; $i < $file\_count; $i++) {

*// get the uploaded file info*

$status = $files['error'][$i];

$filename = $files['name'][$i];

$tmp = $files['tmp\_name'][$i];

*// an error occurs*

if ($status !== UPLOAD\_ERR\_OK) {

$errors[$filename] = MESSAGES[$status];

continue;

}

*// validate the file size*

$filesize = filesize($tmp);

if ($filesize > MAX\_SIZE) {

*// construct an error message*

$message = sprintf(

"The file %s is %s which is greater than the allowed size %s",

$filename,

format\_filesize($filesize),

format\_filesize(MAX\_SIZE)

);

$errors[$filesize] = $message;

continue;

}

*// validate the file type*

if (!in\_array(get\_mime\_type($tmp), array\_keys(ALLOWED\_FILES))) {

$errors[$filename] = "The file $filename is allowed to upload";

}

}

if ($errors) {

response(400, [

'success' => false,

'message' => $errors

]);

}

*// move the files*

for ($i = 0; $i < $file\_count; $i++) {

$filename = $files['name'][$i];

$tmp = $files['tmp\_name'][$i];

$mime\_type = get\_mime\_type($tmp);

*// set the filename as the basename + extension*

$uploaded\_file = pathinfo($filename, PATHINFO\_FILENAME) . '.' . ALLOWED\_FILES[$mime\_type];

*// new filepath*

$filepath = UPLOAD\_DIR . '/' . $uploaded\_file;

*// move the file to the upload dir*

$success = move\_uploaded\_file($tmp, $filepath);

if (!$success) {

$errors[$filename] = "The file $filename was failed to move.";

}

}

if ($errors) {

response(400, [

'success' => false,

'message' => $errors

]);

}

response(200, [

'success' => true,

'message' => 'The files uploaded successfully'

]);

*/\*\**

*\* Return a mime type of file or false if an error occurred*

*\**

*\* @param string $filename*

*\* @return string | bool*

*\*/*

function get\_mime\_type(string $filename)

{

$info = finfo\_open(FILEINFO\_MIME\_TYPE);

if (!$info) {

return false;

}

$mime\_type = finfo\_file($info, $filename);

finfo\_close($info);

return $mime\_type;

}

*/\*\**

*\* Return a human-readable file size*

*\**

*\* @param int $bytes*

*\* @param int $decimals*

*\* @return string*

*\*/*

function format\_filesize(int $bytes, int $decimals = 2): string

{

$units = 'BKMGTP';

$factor = floor((strlen($bytes) - 1) / 3);

return sprintf("%.{$decimals}f", $bytes / pow(1024, $factor)) . $units[(int)$factor];

}

*/\*\**

*\* Response JSON to the client*

*\* @param int $status\_code*

*\* @param array|null $data*

*\*/*

function response(int $status\_code, array $data = null)

{

header("HTTP/1.1 " . $status\_code . " " . HTTP\_STATUSES[$status\_code]);

header("Content-Type: application/json");

echo json\_encode($data);

exit;

}

Code language: JavaScript (javascript)

Read more on [how to upload multiple files in PHP](https://www.phptutorial.net/php-tutorial/php-upload-multiple-files/) for more information.

## **Summary**

* Use the JavaScript FileReader API to read the files selected by users via drag & drop or file input element.

# JavaScript Geolocation

**Summary**: in this tutorial, you’ll learn how to use the Geolocation API to allow web applications to access your location once you agree to share.

## **What is Geolocation API**

The Geolocation API allows the web application to access your location if you agree to share it.

## **Why Geolocation API**

The Geolocation API is useful for web applications that need to work based on the user’s locations such as Search Engines, eCommerce websites, maps, and weather apps.

For example, an eCommerce website can request you to share your location. Once having the information, it can show you the product availability as well as prices and discounts based on your location.

Similarly, the search engine like Google.com can return local search results based on your search term and location.

For example, if you’re in San Fransisco and search for Pizza, Google will show you a list of Pizza restaurants that are near to your current location.

## **The geolocation object**

The Geolocation API is available through the navigator.geolocation object.

### **Check if the Geolocation API is supported**

To check if a browser supports it, you can test if the geolocation property exists on the navigator object like this:

if (!navigator.geolocation) {

console.error(`Your browser doesn't support Geolocation`);

}

Code language: JavaScript (javascript)

### **Get the current position**

To get the user’s current location, you call the getCurrentPosition() method of the navigator.geolocation object.

The getCurrentPosition() method sends an asynchronous request to detect the user’s location and queries the positioning hardware (like GPS) to get the up-to-date data.

The getCurrentPosition() can succeed or fail. It accepts two optional callbacks that will execute in each case respectively.

If the getCurrentPosition() succeeds, you’ll get the user’s position as an GeolocationCoordinates object.

The GeolocationCoordinates object has the latitude and longitude properties that represent a location.

## **JavaScript Geolocation example**

The following example shows a [simple application that has one button](https://www.javascripttutorial.net/sample/webapis/geolocation/). When you click the button, the browser will request you to share your location. If you agree, it’ll show your current latitude and longitude.

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript Geolocation Demo**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<div class="container">**

**<button id="show" class="btn">**Show my location**</button>**

**<div id="message"></div>**

**</div>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

And the following shows the app.js file:

(() => {

const message = document.querySelector('#message');

*// check if the Geolocation API is supported*

if (!navigator.geolocation) {

message.textContent = `Your browser doesn't support Geolocation`;

message.classList.add('error');

return;

}

*// handle click event*

const btn = document.querySelector('#show');

btn.addEventListener('click', function () {

*// get the current position*

navigator.geolocation.getCurrentPosition(onSuccess, onError);

});

*// handle success case*

function onSuccess(position) {

const {

latitude,

longitude

} = position.coords;

message.classList.add('success');

message.textContent = `Your location: (${latitude},${longitude})`;

}

*// handle error case*

function onError() {

message.classList.add('error');

message.textContent = `Failed to get your location!`;

}

})();

Code language: JavaScript (javascript)

How it works.

First, select the #message element by using the querySelector() method.

Next, show an error message if the browser doesn’t support the Geolocation API.

Then, select the button and attach a click event handler. The click event handler calls the getCurrentPosition() method of the navigator.geolocation object to get the current position of the user.

The onSuccess and onError callback functions are for handling the success and error cases.

After that, define the onSuccess() function that gets the latitude and longitude from the position.coords and shows them on the <div> element with id message.

Finally, define the onError() function that shows an error message in case the getCurrentPosition() method fails.

Note that all the code in the app.js are wrapped in an [IIFE](https://www.javascripttutorial.net/javascript-immediately-invoked-function-expression-iife/).

## **Test the app**

The following illustrates how to run the web application using Google Chrome. If you’re using a different browser like Firefox, Edge, or Safari, you should be able to follow the steps.

When you click the **Show my location** button, the browser will request you to share your location like this:

Graphical user interface, application

Description automatically generated

If you click the **Allow** button, you’ll see your location as shown in the following picture:

Graphical user interface, text, application, chat or text message

Description automatically generated

Notice that your location may be different.

To change the browser setting, you click the **location**icon and the **Manage**button:

Graphical user interface, application, Word

Description automatically generated

It’ll show a screen like this:

Graphical user interface, text, application

Description automatically generated

If you click the **Trash**icon, the allowed site will be removed.

And if you [refresh the app](https://www.javascripttutorial.net/sample/webapis/geolocation/) and click the **Show my location** button, it’ll request you to share your location again.

This time if you click the **Block**button:

Graphical user interface, application

Description automatically generated

…the app will show an error message:

Graphical user interface, text, application, chat or text message

Description automatically generated

## **Summary**

* The Geolocation API allows web applications to access the user’s current position.
* Use the navigator.geolocation.getCurrentPosition() method to get the current user’s location.

# JavaScript Notification

**Summary**: in this tutorial, you’ll learn how to use the JavaScript Notification API to show desktop notifications to the users.

The Notification API allows you to use JavaScript to display desktop notifications to the users.

## **Notification permissions**

Since the Notification API can be easy to abuse, it strictly enforces two security features by default:

* First, users must explicitly agree to receive the notifications on a per-origin basis.
* Second, only code runs in a secure context (https) can send the notifications.

When you request users for notification permission, they may agree or deny it.

If users deny it explicitly, the browsers will remember the choice. And you have no second chance to request permission again.

If users don’t agree or refuse i.e., they ignore it, you can send a request for the notification permission again.

To request notification permission, you use the Notification global object. This object has the requestPermission() method that returns a [Promise](https://www.javascripttutorial.net/es6/javascript-promises/), which resolves when the user takes an action on the permission dialog box:

let permission = await Notification.requestPermission();

Code language: JavaScript (javascript)

The permission can be one of the following string 'granted', 'denied', or 'default':

* 'granted' – users accept to receive the notifications.
* 'denined' – users deny to have the notifications.
* 'default' – users choices are unknown and browsers will act as if the values were 'denied'.

## **Show and hide notifications**

To create a notification, you use the Notification constructor. The following creates a simple notification with a title:

const greeting = new Notification('Hi, How are you?');

Code language: JavaScript (javascript)

The notification can be highly customizable with the second options parameter.

For example, the following creates and shows a notification with a body text with an icon:

const greeting = new Notification('Hi, How are you?',{

body: 'Have a good day',

icon: './img/goodday.png'

});

Code language: JavaScript (javascript)

To close a notification, you call the close() method of the Notification object returned from the Notification constructor:

greeting.close();

Code language: CSS (css)

To close the notification after a period of time, you use the [setTimeout()](https://www.javascripttutorial.net/javascript-bom/javascript-settimeout/) function. For example, the following shows how to close the greeting notification after 10 seconds:

setTimeout(() => greenting.close(), 10\*1000);

Code language: JavaScript (javascript)

## **Notification events**

The Notification object provides you with the following events:

* show – is fired when the notification is displayed.
* click – is fired when the notification is clicked.
* close – is fired when the notification is dismissed or closed via the close() method.
* error – is fired when an error occurs, which blocks the notification from being displayed.

To handle these events, you use the [addEventListener()](https://www.javascripttutorial.net/dom/events/add-an-event-handler/) method of the Notification object.

The following example navigates to the URL https://www.javascripttutorial.net/ when the notification is clicked:

*// create a notification object*

const greeting = new Notification('Hi, How are you?',{

body: 'Have a good day',

icon: './img/goodday.png'

});

*// navigate to the https://www.javascripttutorial.net/ on click*

greeting.addEventListener('click', function(){

window.open('https://www.javascripttutorial.net/web-apis/javascript-notification/');

});

Code language: JavaScript (javascript)

Besides using the addEventListener(), you can assign an event handler to the onclick property of the Notification object. For example:

greeting.onclick = () => window.open('https://www.javascripttutorial.net/web-apis/javascript-notification/');

Code language: JavaScript (javascript)

The Notification object has onshow, onclick, onclose and onerror for the corresponding events.

## **JavaScript Notification API example**

In this example, we’ll build [a simple web application that shows a desktop notification to the users.](https://www.javascripttutorial.net/sample/webapis/notification/)

### **Create the project structure**

First, create a new folder called notification, and three subfolders js, css, and img inside the notification folder.

Second, create style.css in the css folder, app.js in the js folder, and index.html in the notification folder.

Third, download and copy the following icon to the img folder. You’ll use it as the icon of the notification.

Logo, icon

Description automatically generated

### **Create the index.html page**

In the index.html page, you place the links to the style.css and app.js files:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript Notification API**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<div class="container">**

**<h1>**JavaScript Notification API Demo**</h1>**

**<div class="error"></div>**

**</div>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The index.html has a heading one and a <div> element that will show an error message if the notification permission is not granted.

### **Make the app.js file**

Since you’ll use the await keyword for calling the Notification.requestPermission() method, you need to place all the code in the app.js in an async [IIFE](https://www.javascripttutorial.net/javascript-immediately-invoked-function-expression-iife/):

(async () => {

*// place js code here*

})();

Code language: JavaScript (javascript)

First, define a function that creates and shows a notification, closes it after 10 seconds, and opens the URL (https://www.javascripttutorial.net/web-apis/javascript-notification/) if it is clicked:

*// create and show the notification*

const showNotification = () => {

*// create a new notification*

const notification = new Notification('JavaScript Notification API', {

body: 'This is a JavaScript Notification API demo',

icon: './img/js.png',

vibrate: true

});

*// close the notification after 10 seconds*

setTimeout(() => {

notification.close();

}, 10 \* 1000);

*// navigate to a URL*

notification.addEventListener('click', () => {

window.open('https://www.javascripttutorial.net/web-apis/javascript-notification/', '\_blank');

});

}

Code language: JavaScript (javascript)

Second, define another function to show an error message if the notification is not granted:

*// show an error message*

const showError = () => {

const error = document.querySelector('.error');

error.style.display = 'block';

error.textContent = 'You blocked the notifications';

}

Code language: JavaScript (javascript)

Third, check if the notification permission is granted. If the user did not take any action, request it.

If the notification permission is granted, then the granted flag is true. Otherwise, it’s false. The showNotification() or showError() function is called based on the value of the granted flag:

let granted = false;

if (Notification.permission === 'granted') {

granted = true;

} else if (Notification.permission !== 'denied') {

let permission = await Notification.requestPermission();

granted = permission === 'granted' ? true : false;

}

*// show notification or the error message*

granted ? showNotification() : showError();

Code language: JavaScript (javascript)

The following shows the complete code of the app.js file:

(async () => {

*// create and show the notification*

const showNotification = () => {

*// create a new notification*

const notification = new Notification('JavaScript Notification API', {

body: 'This is a JavaScript Notification API demo',

icon: './img/js.png'

});

*// close the notification after 10 seconds*

setTimeout(() => {

notification.close();

}, 10 \* 1000);

*// navigate to a URL when clicked*

notification.addEventListener('click', () => {

window.open('https://www.javascripttutorial.net/web-apis/javascript-notification/', '\_blank');

});

}

*// show an error message*

const showError = () => {

const error = document.querySelector('.error');

error.style.display = 'block';

error.textContent = 'You blocked the notifications';

}

*// check notification permission*

let granted = false;

if (Notification.permission === 'granted') {

granted = true;

} else if (Notification.permission !== 'denied') {

let permission = await Notification.requestPermission();

granted = permission === 'granted' ? true : false;

}

*// show notification or error*

granted ? showNotification() : showError();

})();

Code language: JavaScript (javascript)

Here is the [demo page](https://www.javascripttutorial.net/sample/webapis/notification/).

When you open the page, it’ll request notification permission:

Graphical user interface, text, application

Description automatically generated

If you click the allow button, you’ll see the following notification on your desktop:

Text

Description automatically generated

It’ll close in 10 seconds. If you click the notification, it’ll open the URL https://www.javascripttutorial.net/web-apis/javascript-notification/

## **Summary**

* The JavaScript Notification API allows you to show desktop notifications to users.
* The notifications must be explicitly granted by users on an origin basis and only be triggered by the code executing in a secure context (https).
* Use the Notification constructor to create and show notifications.
* Use the Notification events to make the notifications more interactive.

# JavaScript Canvas

**Summary**: in this tutorial, you’ll learn about HTML Canvas and how to use JavaScript to draw on the canvas.

## **Introduction to the HTML5 Canvas element**

HTML5 features the <canvas> element that allows you to draw 2D graphics using JavaScript.

The <canvas> element requires at least two attributes: width and height that specify the size of the canvas:

**<canvas width="500" height="300" id="canvas"></canvas>**

Code language: HTML, XML (xml)

Like other elements, you can access the width and height properties of the <canvas> element via its DOM properties:

const canvas = document.querySelector('#canvas');

const width = canvas.width;*// 500*

const height = canvas.height;*// 300*

Code language: JavaScript (javascript)

And you can also change the width and height of the <canvas> element using the DOM methods:

canvas.width = 600;

canvas.height = 400;

### **Fallback content**

Unlike the <img> element, The <canvas> element requires the closing tag </canvas>. Any content between the opening and closing tags is fallback content that will display only if the browser doesn’t support the <canvas> element. For example:

**<canvas width="500" height="300" id="canvas">**The browser doesn't support the canvas element**</canvas>**

Code language: HTML, XML (xml)

Nowadays, most modern web browsers support the <canvas> element.

### **The rendering context**

Initially, the canvas is blank. To draw something, you need to access the rendering context and use it to draw on the canvas.

The <canvas> element features the getContext() method that returns a render context object.

The getContext() takes one argument which is the type of context. For example, you use the "2d" to get a 2D rendering context object, which is an instance of the CanvasRenderingContext2D interface.

The 2D rendering context allows you to draw shapes, text, images, and other objects.

The following example shows how to select the canvas element using the querySelector() method and access the drawing context by calling its getContext() method:

let canvas = document.querySelector('#canvas');

let ctx = main.getContext('2d');

Code language: JavaScript (javascript)

### **Check for canvas support**

When using the <canvas> element, it’s important to check if the browser supports the getContext() method. To do it, you use the following code:

let canvas = document.querySelector('#main');

if(canvas.getContext) {

let ctx = main.getContext('2d');

}

Code language: JavaScript (javascript)

## **The 2D context**

The 2D drawing context features methods for drawing simple 2D shapes such as paths, rectangles, and arcs.

The coordinates in a 2D context begin at the upper-left of the <canvas> element, which is point (0,0) as shown in the following picture:

Chart

Description automatically generated

All coordinate values are calculated in relation to the (0,0) with x increasing to the right and y increasing to the bottom.

By default, the width and height determine the number of pixels is available in each direction.

### **Fills and strokes**

Fill and stroke are two basic drawing operation on 2D drawing context.

* Fill fills in the shape with a specific style such as color, gradient, and image.
* Stroke adds colors to the edges of the shape.

The fillStyle and strokeStyle properties of the 2D drawing context will determine the fill and stroke styles.

You can set these properties to a string, a gradient object, or a pattern object. By default, they both set to a value of '#000000'.

The following illustrates how to set the fill and stroke styles for the 2D drawing context and draw a rectangle.

(() => {

const canvas = document.querySelector('#main');

if (!canvas.getContext) {

return;

}

*// get the context*

let ctx = canvas.getContext('2d');

*// set fill and stroke styles*

ctx.fillStyle = '#F0DB4F';

ctx.strokeStyle = 'red';

*// draw a rectangle with fill and stroke*

ctx.fillRect(50, 50, 150, 100);

ctx.strokeRect(50, 50, 150, 100);

})();

Code language: JavaScript (javascript)

[Click here to see the canvas demo](https://www.javascripttutorial.net/sample/webapis/canvas/).

Note that you’ll learn how to draw a rectangle in a later tutorial.

## **Summary**

* Use the HTML5 canvas element for drawing 2D graphics.
* Use the getContext('2d') to get the 2D drawing context for drawing 2D graphics on canvas.
* Use the fillStyle and StrokeStyle properties to set the styles for the 2D drawing context.

# JavaScript fillRect

**Summary**: in this tutorial, you’ll learn how to use the JavaScript fillRect() method to draw rectangles with a specified width and height on a canvas.

## **Introduction to the JavaScript fillRect() method**

The fillRect() is a method of the 2d drawing context object. The fillRect() method allows you to draw a filled rectangle at (x,y) position with a specified with and height on a [canvas](https://www.javascripttutorial.net/web-apis/javascript-canvas/).

The following shows the syntax of the fillRect() method:

ctx.fillRect(x,y,width,height);

Code language: CSS (css)

In this syntax:

* x is the x-axis coordinate of the starting point of the rectangle.
* y is the y-axis coordinate of the starting point of the rectangle.
* width is the rectangles’ width. It can be positive or negative. The positive values are to the right while the negative values are to the left.
* height is the rectangle’s height. It can be also positive or negative. The positive values are down while the negative values are up.

The fillStyle attribute of the 2d drawing context will determine the fill style of the rectangle.

## **JavaScript fillRect() example**

The following shows the index.html file that has a canvas element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript Canvas fillRect**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<h1>**JavaScript Canvas fillRect**</h1>**

**<canvas id="canvas" height="400" width="500">**

**</canvas>**

**<script src="js/index.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The following drawRectangles() function in the app.js file will draw two rectangles:

function drawRectangles() {

const canvas = document.querySelector('#canvas');

if (!canvas.getContext) {

return;

}

const ctx = canvas.getContext('2d');

ctx.fillStyle = '#F9DC5C';

ctx.fillRect(100, 100, 150, 100);

ctx.fillStyle = 'rgba(0,0,255,0.5)';

ctx.fillRect(200, 150, -150, -100);

}

drawRectangles();

Code language: JavaScript (javascript)

How it works.

First, select the [canvas element](https://www.javascripttutorial.net/web-apis/javascript-canvas/) using the [document.querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method:

const canvas = document.querySelector('#canvas');

Code language: JavaScript (javascript)

Second, check if the browser supports the canvas API:

if (!canvas.getContext) {

return;

}

Code language: JavaScript (javascript)

Third, get the 2d drawing context object:

const ctx = canvas.getContext('2d');

Code language: JavaScript (javascript)

Fourth, set the fill style to the #F9DC5C color and draw the first rectangle using the fillRect() method:

ctx.fillStyle = '#F9DC5C';

ctx.fillRect(100, 100, 150, 100);

Code language: JavaScript (javascript)

The first rectangle starts at (100,100) and has the width of 150 pixels and the height of 100 pixels.

Finally, set the fill style to blue with the alpha 0.5, which creates a transparent effect. And use the fillRect() method to draw the second rectangle:

ctx.fillStyle = 'rgba(0,0,255,0.5)';

ctx.fillRect(200, 150, -150, -100);

Code language: JavaScript (javascript)

The second rectangle starts at (200,150). Since we passed the negative width and height to the fillRect() method, the width is to the left and the height is to up.

The following picture shows the output:

A picture containing logo

Description automatically generated

[Here is the link to the demo](https://www.javascripttutorial.net/sample/webapis/fillrect/).

## **Using the JavaScript fillRect() to draw a brick wall**

In this example, you’ll learn how to use the fillRect() method to draw a brick wall.

The following illustates the index.html file that has a canvas element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript Canvas fillRect**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<h1>**JavaScript fillRect() - Draw a Brick Wall**</h1>**

**<canvas id="canvas" height="400" width="500">**

**</canvas>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this example, the canvas will have the following style specified in the style.css file:

canvas {

border: solid 1px #ccc;

background-color: #000;

}

Code language: CSS (css)

In the app.js file, define a function called drawWall() that draws a brick wall on a canvas:

function drawWall(ctx, canvasWidth, canvasHeight) {

*// define brick height and width, and spaces between them*

const bh = 20,

bw = 50,

space = 5;

*// calculate the rows and columns of the wall*

const rows = Math.ceil(canvasHeight / (bh + space));

const columns = Math.ceil(canvasWidth / (bw + space));

*// draw columns*

for (let r = 0; r < rows; r++) {

*// draw rows*

for (let c = 0; c < columns; c++) {

if (r % 2) {

c == 0 ? ctx.fillRect(c \* (bw + space), r \* (bh + space), bw / 2, bh) :

ctx.fillRect(c \* (bw + space) - bw / 2, r \* (bh + space), bw, bh);

} else {

ctx.fillRect(c \* (bw + space), r \* (bh + space), bw, bh);

}

}

}

}

Code language: JavaScript (javascript)

The drawWall() function accepts a 2d drawing context and the height and width of the wall.

First, define the height (bh) and width (bw) of each brick and the space between two bricks:

const bh = 20,

bw = 50,

space = 5;

Code language: JavaScript (javascript)

Second, calculate the number of rows and columns of bricks based on the dimension of the brick and the wall’s width and height:

*// calculate the rows and columns of the wall*

const rows = Math.ceil(height / (bh + space));

const columns = Math.ceil(width / (bw + space));

Code language: JavaScript (javascript)

Third, draw bricks by using the fillRect() method. Use two [for loops](https://www.javascripttutorial.net/javascript-for-loop/) to draw bricks row by row.

*// draw rows*

for (let r = 0; r < rows; r++) {

*// draw columns*

for (let c = 0; c < columns; c++) {

if (r % 2) {

c == 0 ? ctx.fillRect(c \* (bw + space), r \* (bh + space), bw / 2, bh) :

ctx.fillRect(c \* (bw + space) - bw / 2, r \* (bh + space), bw, bh);

} else {

ctx.fillRect(c \* (bw + space), r \* (bh + space), bw, bh);

}

}

}

Code language: JavaScript (javascript)

Note that the first brick of the even row will have a width that equals to a half of the normal brick’s width.

The following picture shows the output:

A picture containing text

Description automatically generated

And here is the link shows the demo.

## **Summary**

* Use the JavaScript fillRect() to draw a filled rectangle that starts at (x,y) and has a specified width and height.
* Use the fillStyle property to set the fill style of the rectangle.

# JavaScript strokeRect

**Summary**: in this tutorial, you’ll learn how to use the JavaScript strockRect() method to draw an outlined rectangle on a canvas.

## **Introduction to the JavaScript strokeRect() method**

The strokeRect() is a method of the 2D drawing context. The strokeRect() allows you to draw an outlined rectangle with the stroke style derived from the current strokeStyle property.

The following shows the syntax of the strokeRect() method:

ctx.strokeRect(x, y, width, height);

Code language: CSS (css)

In this syntax:

* x is the x-axis coordinate of the starting point of the rectangle.
* y is the y-axis coordinate of the starting point of the rectangle.
* width is the rectangle’s width. It can be positive or negative. If the width is positive, the method draws the rectangle from (x,y) to the right. Otherwise, it draws the rectangle from the (x,y) to the left.
* height is the rectangle’s height. And the height can also be positive or negative. If it’s positive, the method draws the rectangle from the (x,y) to the bottom. Otherwise, it draws the rectangle from the (x,y) to the top.

The strokeRect() draws directly to the canvas without modifying the current path. It means that any subsequent fill() or stroke() call will have no effect.

## **The JavaScript strokeRect() example**

The following shows the index.html page that contains a canvas element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript strokeRect**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<h1>**JavaScript strokeRect() Demo**</h1>**

**<canvas id="canvas" height="400" width="500">**

**</canvas>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In the app.js file, define a function that draws two outlined rectangles:

function drawOutlinedRects() {

const canvas = document.querySelector('#canvas');

if (!canvas.getContext) {

return;

}

const ctx = canvas.getContext('2d');

ctx.strokeStyle = 'red';

ctx.strokeRect(100, 100, 150, 100);

ctx.strokeStyle = 'green';

ctx.strokeRect(200, 150, -150, -100);

}

drawOutlinedRects();

Code language: JavaScript (javascript)

The following picture shows the output:

A picture containing shape

Description automatically generated

[Here is the strokeRect() demo.](https://www.javascripttutorial.net/sample/webapis/strokerect/)

How it works.

* First, select the canvas element by using the querySelector() method.
* Next, check if the browser supports the canvas API.
* Then, get the 2D drawing context from the canvas element.
* After that, set the stroke style to red and use the strokeRect() method to draw the first outline rectangle.
* Finally, set the stroke style to green and draw the second outlined rectangle. In this case, we passed the negative width and height to the strokeRect() method to draw the rectangle to the left and top from its starting point.

## **Summary**

* Use the strokeRect() method to draw an outlined rectangle starting at (x, y) with a specified width and height.

# Canvas clearRect

**Summary**: in this tutorial, you’ll learn how to use the clearRect() method to erase an area of the canvas.

## **Introduction to the Canvas clearRect() method**

The clearRect() is a method of the 2D drawing context. The clearRect() method clears an area of the canvas by making that area transparent.

In practice, you draw shapes and then use the clearRect() method to clear specific areas to create some interesting effects.

The following shows the syntax of the clearRect() method:

ctx.clearRect(x, y, width, height);

Code language: CSS (css)

The clearRect() has four parameters that define an area to erase:

* x is the x-axis coordinate of the starting point of the rectangle.
* y is the y-axis coordinate of the starting point of the rectangle.
* width is the rectangle’s width. The sign of the width will determine the direction of the rectangle. Positive values are to the right while the negative values are to the left of the starting point.
* height is the rectangle’s height. It also accepts both positive and negative values. Positive values are down while the negative values are up from the starting point.

## **The canvas clearRect() method example**

The following shows the index.html that contains a canvas element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript Canvas clearRect**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<h1>**JavaScript clearRect() Demo**</h1>**

**<canvas id="canvas" height="400" width="500">**

**</canvas>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In the app.js file, define the draw() that will draw on the canvas:

function draw() {

const canvas = document.querySelector('#canvas');

if (!canvas.getContext) {

return;

}

const ctx = canvas.getContext('2d');

*// draw two squares*

ctx.fillStyle = '#F9DC5C';

ctx.fillRect(50, 50, 150, 150);

ctx.fillStyle = 'rgba(0,0,255,0.5)';

ctx.fillRect(100, 100, 150, 150);

*// clear the intersection*

ctx.clearRect(100, 100, 100, 100);

}

draw();

Code language: JavaScript (javascript)

The following picture shows the output:

A picture containing icon

Description automatically generated

And here [is the clearRect() demo](https://www.javascripttutorial.net/sample/webapis/clearrect/).

How it works.

* First, select the canvas element using the [document.querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Next, check if the browser supports the canvas API.
* Then, get the 2D drawing context for drawing on the canvas.
* Then, draw two intersecting squares by using the [fillRect()](https://www.javascripttutorial.net/web-apis/javascript-fillrect/) method. The second square has alpha transparency.
* Finally, clear the intersection of the two squares by using the clearRect() method.

## **Summary**

* Use the clearRect() method to set the pixels in a rectangular area on a canvas to transparent black.

# How to Draw a Line in JavaScript

**Summary**: in this tutorial, you’ll learn how to draw a line using the Canvas API.

## **Steps for drawing a line in JavaScript**

To draw a line on a [canvas](https://www.javascripttutorial.net/web-apis/javascript-canvas/), you use the following steps:

* First, create a new line by calling the beginPath() method.
* Second, move the drawing cursor to the point (x,y) without drawing a line by calling the moveTo(x, y).
* Finally, draw a line from the previous point to the point (x,y) by calling the lineTo(x,y) method.

### **Set the line stroke**

If you want to stroke the line with the strokeStyle, you can call the stroke() method after calling the lineTo(x,y) method.

### **Set the line width**

To set the width for a line, you use the lineWidth property of the 2D drawing context before calling stroke() method:

ctx.lineWidth = 10;

### **The lineTo(x,y) method**

The lineTo(x,y ) method accepts both positive and negative arguments.

If the x is positive, the lineTo(x,y) method draws the line from the starting point to the right. Otherwise, it draws the line from the starting point to the left.

If the y is positive, the lineTo(x,y) method draws the line from the starting point down the y-axis. Otherwise, it draws the line from the starting point up to the y-axis.

## **Drawing a line example**

The following shows the index.html file that contains a canvas element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript - Drawing a Line**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<h1>**JavaScript - Drawing a Line**</h1>**

**<canvas id="canvas" height="400" width="500">**

**</canvas>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

And this app.js contains that draws a line with the color red, 5-pixel width from the point (100, 100) to (300, 100):

function draw() {

const canvas = document.querySelector('#canvas');

if (!canvas.getContext) {

return;

}

const ctx = canvas.getContext('2d');

*// set line stroke and line width*

ctx.strokeStyle = 'red';

ctx.lineWidth = 5;

*// draw a red line*

ctx.beginPath();

ctx.moveTo(100, 100);

ctx.lineTo(300, 100);

ctx.stroke();

}

draw();

Code language: JavaScript (javascript)

The following shows the output:

A picture containing graphical user interface

Description automatically generated

[Here is the link that shows the canvas with the line.](https://www.javascripttutorial.net/sample/webapis/drawline/)

### **Develop a resuable drawLine() function**

The following drawLine() function draws a line from one point to another with a specified stroke and width:

function drawLine(ctx, begin, end, stroke = 'black', width = 1) {

if (stroke) {

ctx.strokeStyle = stroke;

}

if (width) {

ctx.lineWidth = width;

}

ctx.beginPath();

ctx.moveTo(...begin);

ctx.lineTo(...end);

ctx.stroke();

}

Code language: JavaScript (javascript)

To draw a line from (100,100) to (100,300) with the line color green and line width 5 pixels, you can call the drawLine() function as follows:

const canvas = document.querySelector('#canvas');

if (canvas.getContext) {

const ctx = canvas.getContext('2d');

drawLine(ctx, [100, 100], [100, 300], 'green', 5);

}

Code language: JavaScript (javascript)

## **Summary**

* Use beginPath(), moveTo(x, y) and lineTo(x,y) to draw a line.
* Use the strokeStyle and lineWidth to set the line stroke and line width.

# JavaScript fillText

**Summary**: in this tutorial, you’ll learn how to use the JavaScript fillText() method to draw a text string to a canvas.

## **Introduction to the JavaScript fillText() method**

The fillText() is a method of a 2D drawing context. The fillText() method allows you to draw a text string at a coordinate with the fill derived from the current fillStyle.

The following shows the syntax of the fillText() method:

ctx.fillText(text, y , y [, maxWidth])

Code language: CSS (css)

The fillText() accepts the following parameters:

* text is the text string to draw.
* x and y is the x-axis and y-axis coordinates of the point at which the method starts drawing text.
* maxWidth is the maximum number of pixels wide that the method will render the text. By default, the text width will have no limit if you omit the maxWidth argument. However, if you pass the maxWith value, the method will try to adjust the kerning or select a more condensed font for the text to make it fit in the specified width.

## **JavaScript fillText() example**

Let’s take some examples of using the JavaScript fillText() method.

### **1) Draw a filled text example**

[This example](https://www.javascripttutorial.net/sample/webapis/filltext/) draws the words "Hello, Canvas!" on a canvas using the fillText() method.

#### **HTML**

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**JavaScript fillText Demo**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<canvas id="canvas" height="400" width="500">**

**</canvas>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

#### **JavaScript**

The following shows the JavaScript code that draws the text:

const canvas = document.getElementById('canvas');

if (canvas.getContext) {

const ctx = canvas.getContext('2d');

ctx.fillStyle = 'green';

ctx.font = '60px san-serif';

ctx.fillText('Hello, Canvas!', 100, 200);

}

Code language: JavaScript (javascript)

Output:

[Logo, company name

Description automatically generated](https://www.javascripttutorial.net/sample/webapis/filltext/)

How it works.

* First, select the canvas element using the querySelector() method.
* Second, get the reference to the canvas 2D graphics context.
* Third, set the font to 60-pixel-tall san-serif. and the fill style is green.
* Finally, draw the text 'Hello, Canvas!' starting at the coordinates (100,200).

### **2) Constraint the text size**

The following example draws the words 'Hello, Canvas!' with a maximum with of 250px.

#### **HTML**

**<canvas id="canvas" height="400" width="500">**

**</canvas>**

Code language: HTML, XML (xml)

#### **JavaScript**

const canvas = document.getElementById('canvas');

if (canvas.getContext) {

const ctx = canvas.getContext('2d');

ctx.fillStyle = 'green';

ctx.font = '60px san-serif';

ctx.fillText('Hello, Canvas!', 100, 200, 250);

}

Code language: JavaScript (javascript)

Output:

Logo, company name

Description automatically generated

## **Text alignments**

To align the text on the canvas, you use the textAlign property:

ctx.textAlign = value;

The alignment is relative to the x of the fillText() method.

The value can be one of the following values:

* 'left' – the text is left-aligned.
* 'right' – the text is right-aligned.
* 'center' – the text is centered.
* 'start' – the text is aligned at the start of the line. It’s left-aligned in the left-to-right locales and right-aligned in the right-to-left locales.
* 'end' – the text is aligned at the end of the line. It’s right-aligned in the left-to-right locales and left-aligned for the right-to-left locales.

The default value for the texAlign is start.

The [following example](https://www.javascripttutorial.net/sample/webapis/textalign/) demonstrates the various options of the textAlign property:

#### **HTML**

**<canvas id="canvas" height="350" width="500">**

**</canvas>**

Code language: HTML, XML (xml)

#### **JavaScript**

const canvas = document.getElementById('canvas');

const ctx = canvas.getContext('2d');

const x = canvas.width / 2;

ctx.beginPath();

ctx.moveTo(x, 0);

ctx.lineTo(x, canvas.height);

ctx.stroke();

ctx.font = '25px san-serif';

ctx.textAlign = 'left';

ctx.fillText('left-aligned', x, 40);

ctx.textAlign = 'center';

ctx.fillText('center-aligned', x, 85);

ctx.textAlign = 'right';

ctx.fillText('right-aligned', x, 130);

*// LTR locale*

canvas.setAttribute('dir', 'ltr');

ctx.textAlign = 'start';

ctx.fillText('start', x, 175);

ctx.textAlign = 'end';

ctx.fillText('end', x, 220);

*// RTL locale*

canvas.setAttribute('dir', 'rtl');

ctx.textAlign = 'start';

ctx.fillText('start', x, 265);

ctx.textAlign = 'end';

ctx.fillText('end', x, 305);

Code language: JavaScript (javascript)

To change the locale to LTR or RTL, you set the dir attribute value of the canvas to 'ltr' and 'rtl'.

Here is the output:

[A picture containing chart

Description automatically generated](https://www.javascripttutorial.net/sample/webapis/textalign/)

## **Text baseline**

To specify the text baseline for the drawing text, you use the textBaseline property of the 2D drawing context:

ctx.textBaseline = value;

The value of the textBaseline can be one of the following values:

* 'top' – the text baseline is the top of the em square.
* 'hanging' – the text baseline is the hanging baseline.
* 'middle' – the text baseline is the middle of the em square.
* 'alphabetic' – the text baseline is the alphabetic baseline. This is the default value.
* 'ideographic' – the text baseline is ideographic. It’s mainly used by Chinese, Japanese, and Korean scripts.
* 'bottom' – the text baseline is the bottom of the bounding box.

### **Text baseline example**

The [following example](https://www.javascripttutorial.net/sample/webapis/textbaseline/) illustrates various textBaseline values.

#### **HTML**

**<canvas id="canvas" width="550" height="500"></canvas>**

Code language: HTML, XML (xml)

### **JavaScript**

const canvas = document.getElementById('canvas');

if (canvas.getContext) {

const ctx = canvas.getContext('2d');

const baselines = ['top', 'hanging', 'middle', 'alphabetic', 'ideographic', 'bottom'];

const str = 'The quick brown fox jumps over the lazy dog';

ctx.font = '20px san-serif';

ctx.strokeStyle = 'red';

baselines.forEach((baseline, index) => {

*// set the text baseline*

ctx.textBaseline = baseline;

const y = 75 + index \* 75;

*// draw a line*

ctx.beginPath();

ctx.moveTo(0, y + 0.5);

ctx.lineTo(500, y + 0.5);

ctx.stroke();

*// draw the text*

ctx.fillText(`${str}(${baseline})`, 0, y);

});

}

Code language: JavaScript (javascript)

Output:

[Table

Description automatically generated](https://www.javascripttutorial.net/sample/webapis/textbaseline/)

## **Summary**

* Use the JavaScript fillText() to draw a text string at a coordinate to a canvas.
* Use the font, textAlign, and textBaseline property to set the options for drawing text.

# JavaScript Translate

**Summary**: in this tutorial, you’ll learn how to use the JavaScript translate() API to move the origin of the canvas to the new position.

## **Introduction to the JavaScript translate() method.**

The translate() is a method of a 2D drawing context. The translate(x,y) method moves the canvas and its origin x units horizontally and y units vertically.

The following illustrates the syntax of the translate() method:

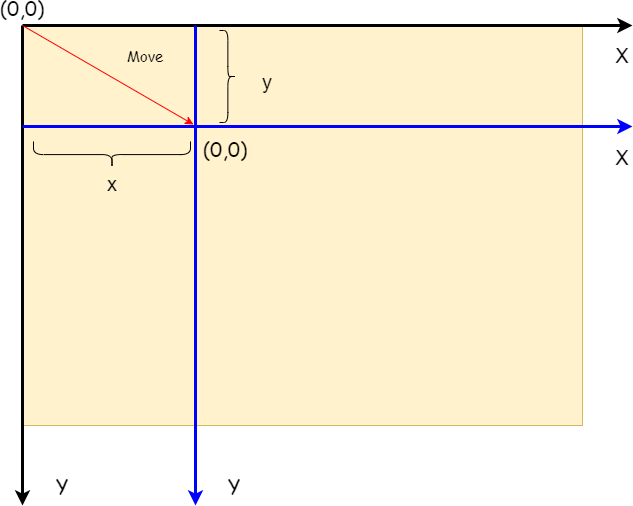
ctx.translate(x, y);

Code language: CSS (css)

In this syntax:

* x represents the distance that you want to move the origin of the canvas in the horizontal direction. The origin moves to the left if x is positive and to the right in case x is negative.
* y represents the distance that you want to move the origin of the canvas in the vertical direction. The origin moves down if y is positive and up if y is negative.

By default, the origin of the canvas (0,0) is at the top left of the screen. By adding a translation transformation, the whole coordinate system moves so that its new origin locates at (x,y):



The translate() can be very useful in drawing. Suppose you want to draw two objects, one is the translation of another.

To do it, you can draw the first object, apply a translation transformation, and draw the second object.

If you don’t apply the translation transformation, you need to calculate the new coordinates for the second object.

## **JavaScript translate() examples**

Let’s take some examples of using the JavaScript translate() method.

### **1) Simple JavaScript translate() example**

The following example draws a square at (100,100) and the second square at (150,150). It calls the translate() method to move the original origin of the canvas to (150, 150) to draw the second square.

#### **HTML**

**<canvas id="canvas" height="300" width="500">**

**</canvas>**

Code language: HTML, XML (xml)

### **JavaScript**

const canvas = document.querySelector('#canvas');

if (canvas.getContext) {

const ctx = canvas.getContext('2d');

*// draw the first square*

ctx.fillStyle = 'red';

ctx.fillRect(100, 100, 50, 50);

*// translate*

ctx.translate(150, 150);

*// draw the second square*

ctx.fillStyle = 'blue';

ctx.fillRect(0, 0, 50, 50);

}

Code language: JavaScript (javascript)

Output:

Chart, waterfall chart

Description automatically generated

### **2) Using JavaScript translate() to draw a clock**

[The following example draws a clock at the center of the canvas](https://www.javascripttutorial.net/sample/webapis/translate/). To make it easy to draw the hour and minute hands, it translates the origin of the canvas to the center of the clock.

#### **HTML**

**<canvas id="canvas" height="300" width="500">**

**</canvas>**

Code language: HTML, XML (xml)

#### **JavaScript**

const canvas = document.querySelector('#canvas');

if (canvas.getContext) {

const ctx = canvas.getContext('2d');

ctx.beginPath();

const centerX = canvas.width / 2,

centerY = canvas.height / 2;

*// draw the circle*

ctx.arc(centerX, centerY, 70, 0, 2 \* Math.PI, false);

*// translate to center*

ctx.translate(centerX, centerY);

*// draw the hour hand*

ctx.moveTo(0, 0);

ctx.lineTo(-30, -20);

*// draw the minute hand*

ctx.moveTo(0, 0);

ctx.lineTo(0, -55);

ctx.stroke();

}

Code language: JavaScript (javascript)

[Here is the demo link](https://www.javascripttutorial.net/sample/webapis/translate/).

## **Summary**

* Use the JavaScript translate() to move the canvas and its origin x units horizontally and y units vertically.

# JavaScript Rotate

**Summary**: in this tutorial, you’ll learn how to use the JavaScript rotate() method to rotate drawing objects.

## **Introduction to JavaScript rotate() canvas API**

The rotate() is a method of the 2D drawing context. The rotate() method allows you to rotate a drawing object on the canvas.

Here is the syntax of the rotate() method:

ctx.rotate(angle)

Code language: CSS (css)

The rotate() method accepts a rotation angle in radians.

If the angle is positive, the rotation is clockwise. In case the angle is negative, the rotation is counterclockwise.

To convert a degree to a radian, you use the following fomular:

degree \* Math.PI / 180

Code language: JavaScript (javascript)

When adding a rotation, the rotate() method uses the canvas origin as the rotation center point.

The following picture illustrates the rotation:

Diagram

Description automatically generated with medium confidence

If you want to change the rotation center point, you need to move the origin of the canvas using the [translate()](https://www.javascripttutorial.net/web-apis/javascript-translate/) method.

## **JavaScript rotate() example**

[The following example](https://www.javascripttutorial.net/sample/webapis/rotate/) draws a red rectangle starting from the center of the canvas. It then translates the origin of the canvas to the canvas’ center and draws the second rectangle with a rotation of 45 degrees:

### **HTML**

**<canvas id="canvas" height="300" width="500">**

**</canvas>**

Code language: HTML, XML (xml)

### **JavaScript**

const canvas = document.querySelector('#canvas');

if (canvas.getContext) {

*// rectangle's width and height*

const width = 150,

height = 20;

*// canvas center X and Y*

const centerX = canvas.width / 2,

centerY = canvas.height / 2;

const ctx = canvas.getContext('2d');

*// a red rectangle*

ctx.fillStyle = 'red';

ctx.fillRect(centerX, centerY, width, height);

*// move the origin to the canvas' center*

ctx.translate(centerX, centerY);

*// add 45 degrees rotation*

ctx.rotate(45 \* Math.PI / 180);

*// draw the second rectangle*

ctx.fillStyle = 'rgba(0,0,255,0.5)';

ctx.fillRect(0, 0, width, height);

}

Code language: JavaScript (javascript)

[Here is the demo link.](https://www.javascripttutorial.net/sample/webapis/rotate/)

Output:

Logo, company name

Description automatically generated

## **Summary**

* Use JavaScript rotate() method to rotate a drawing object on a canvas.

# JavaScript Scale

**Summary**: in this tutorial, you’ll learn how to use the JavaScript scale() Canvas API to scale drawing objects effectively.

## **Introduction to JavaScript scale() Canvas API**

The scale() is a method of the 2D drawing context. The scale() method adds a scaling transformation to the canvas units horizontally and/or vertically.

The default unit on the canvas is one pixel. The scaling transformation changes this default.

For example, a scaling factor of 0.5 will change the unit size to 0.5 pixels. Similarly, a scaling factor of 2 will increase the unit size to two pixels.

When the unit size is scaled, shapes are also drawn at the new units.

The following illustrates the scale() method:

ctx.scale(x, y)

Code language: CSS (css)

The scale() method accepts two parameters:

* x is the scaling factor in the horizontal direction.
* y is the scaling factor in the vertical direction.

## **JavaScript scale() examples**

The [following example](https://www.javascripttutorial.net/sample/webapis/scale/) first draws a rectangle starting at (10,10) with a width of 100 pixels and a height of 50 pixels. It then adds a scaling factor of 2 to the canvas both vertically and horizontally. Finally, it draws the rectangle with the same width and height. However, the second rectangle is four times bigger than the first one because of the scaling transformation.

### **HTML**

**<canvas id="canvas" height="300" width="500">**

**</canvas>**

Code language: HTML, XML (xml)

### **JavaScript**

const canvas = document.querySelector('#canvas');

if (canvas.getContext) {

*// rectangle width and height*

const width = 100,

height = 50;

*// starting point*

const x = 10,

y = 10;

const ctx = canvas.getContext('2d');

ctx.strokeStyle = 'red';

ctx.strokeRect(x, y, width, height);

ctx.scale(2, 2);

ctx.strokeStyle = 'rgba(0,0,255,.2)';

ctx.strokeRect(x / 2, y / 2, width, height);

}

Code language: JavaScript (javascript)

Output:

[Graphical user interface, application

Description automatically generated](https://www.javascripttutorial.net/sample/webapis/scale/)

[Here is the demo link](https://www.javascripttutorial.net/sample/webapis/scale/).

## **Summary**

* Use the JavaScript scale() to add a scaling transformation to the canvas units horizontally and/or vertically

# JavaScript History pushState

**Summary**: in this tutorial, you’ll learn how to use the JavaScript history pushState() method.

## **Introduction to the JavaScript history pushState() method**

The history.pushState() method allows you to add an entry to the [web browser’s session history stack](https://www.javascripttutorial.net/javascript-bom/javascript-history/).

Here’s the syntax of the pushState() method:

history.pushState(state, title, [,url])

Code language: CSS (css)

The pushState() method accepts three parameters:

### **1) state**

The state is a serializable object. When you navigate to a new state, a popstate event is fired. And the popstate event has a state property that references the history entry’s state object.

### **2) title**

Most browser currently ingore this title property. If you want to change the title of the document, you can use the documen.title property instead.

In practice, you pass an empty string to the title parameter.

### **3) url**

The optional url allows you to define the new history entry’s URL. The URL must be the same origin as the current URL or the method will throw an exception.

When you set the new url, the web browser won’t load the url. The url defaults to the current URL if you don’t specify it.

## **JavaScript history pushState() example**

We’ll make a simple application that shows three tabs: React, Vue, and Angular.

When you click a tab, it’ll show the content of the selected tab. it’ll also update the URL using the history.pushState() method:

[Graphical user interface, text, application, chat or text message

Description automatically generated](https://www.javascripttutorial.net/wp-content/uploads/2021/01/javascript-history-pushstate.gif)

If you copy the URL with the hashtag and load it from the web browser, the app will load the corresponding content associated with that URL.

[Click the following link to see the demo of the app.](https://www.javascripttutorial.net/sample/webapis/history/index.html)

### **Make the index.html page**

The following defines the index.html page:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8" />**

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

**<title>**JavaScript History API: pushState Demo**</title>**

**<link rel="stylesheet" href="css/style.css" />**

**</head>**

**<body>**

**<div class="container">**

**<div class="tabs">**

**<ul>**

**<li class="active" id="tab1">**React**</li>**

**<li id="tab2">**Vue**</li>**

**<li id="tab3">**Angular**</li>**

**</ul>**

**<div class="content">**

A JavaScript library for building user interfaces

**</div>**

**</div>**

**</div>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

And you can find the [style.css](https://www.javascripttutorial.net/sample/webapis/history/css/style.css) file here.

In the app.js file:

First, select the tabs and content elements using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method:

const tabs = document.querySelector(".tabs");

const content = document.querySelector(".tabs > .content");

Code language: JavaScript (javascript)

Second, define a [map](https://www.javascripttutorial.net/es6/javascript-map/) object that associates the URL hash with each tab id:

const hashes = new Map([

["#react", "tab1"],

["#vue", "tab2"],

["#angular", "tab3"],

]);

Code language: JavaScript (javascript)

Third, define another map called data for mapping the tab id with an object. The object has two properties: url and content.

const data = new Map([

[

"tab1",

{

url: "index.html#react",

content:

"React is a JavaScript library for building user interfaces.",

},

],

[

"tab2",

{

url: "index.html#vue",

content: "Vue is the Progressive JavaScript Framework.",

},

],

[

"tab3",

{

url: "index.html#angular",

content:

"Angular is a platform for building mobile and desktop web applications.",

},

],

]);

Code language: JavaScript (javascript)

Fourth, when each tab (or li element) is clicked, the click event occurs. To make it more efficient, we’ll use the [event delegation](https://www.javascripttutorial.net/javascript-dom/javascript-event-delegation/).

So instead of handling the click event on each tab, we’ll handle the click event on the parent of each tab:

tabs.addEventListener("click", function (event) {

if (!event.target.id) return;

update(event.target.id);

});

Code language: JavaScript (javascript)

The if statement ensures that the event handler only updates the content and url if the click event occurs on each individual tab. It’ll do nothing when you click the content area of the tab.

Inside the event handler, we call the update() function and pass the tab id into it.

Fifth, the following defines the update() function:

const update = (tabId) => {

*// remove the active class of the previously selected tab*

const currentTab = tabs.querySelector(".active");

if (currentTab.id != tabId) {

currentTab.classList.remove("active");

}

*// add active class to the selected tab*

const selectedTab = document.getElementById(tabId);

selectedTab.classList.add("active");

const entry = data.get(tabId);

if (entry) {

*// update the URL*

history.pushState(null, "", entry.url);

*// change the content*

content.innerHTML = entry.content;

}

};

Code language: JavaScript (javascript)

The update() function removes the .active class from the current tab and sets the same CSS class to the currently selected tab.

It also gets the url and content from the data based on the tab id. To update the URL, it uses the history.pushState() method.

The app should be working as expected with only one issue.

If you copy the URL:

https:*//www.javascripttutorial.net/sample/webapis/history/index.html#angular*

Code language: JavaScript (javascript)

… and paste it into the new browser window, the app will show the React tab instead of Angular tab.

To fix this, we get the hash from the URL using the [location](https://www.javascripttutorial.net/javascript-bom/javascript-location/) object and call the update() function when the page is loaded.

(() => {

*// get tab id from the hash*

const tabId = hashes.get(window.location.hash);

*// update the tab*

if (tabId) update(tabId);

})();

Code language: JavaScript (javascript)

The following shows the complete app.js file:

const tabs = document.querySelector(".tabs");

const content = document.querySelector(".tabs > .content");

*// store the relationship between hash & tab id*

const hashes = new Map([

["#react", "tab1"],

["#vue", "tab2"],

["#angular", "tab3"],

]);

*// store the relationship between tab id and contents*

const data = new Map([

[

"tab1",

{

url: "index.html#react",

content:

"React is a JavaScript library for building user interfaces.",

},

],

[

"tab2",

{

url: "index.html#vue",

content: "Vue is the Progressive JavaScript Framework.",

},

],

[

"tab3",

{

url: "index.html#angular",

content:

"Angular is a platform for building mobile and desktop web applications.",

},

],

]);

tabs.addEventListener("click", function (event) {

if (!event.target.id) return;

update(event.target.id);

});

const update = (tabId) => {

*// remove the active class of the previously selected tab*

const currentTab = tabs.querySelector(".active");

if (currentTab.id != tabId) {

currentTab.classList.remove("active");

}

*// add active class to the selected tab*

const selectedTab = document.getElementById(tabId);

selectedTab.classList.add("active");

const entry = data.get(tabId);

if (entry) {

*// update the URL*

history.pushState(null, "", entry.url);

*// change the content*

content.innerHTML = entry.content;

}

};

(() => {

*// get tab id from the hash*

const tabId = hashes.get(window.location.hash);

*// update the tab*

if (tabId) update(tabId);

})();

Code language: JavaScript (javascript)

## **Summary**

* Use the history.pushState() method to add an entry to the web browser’s session history stack.

# JavaScript Fetch API

**Summary**: in this tutorial, you’ll learn about the JavaScript Fetch API and how to use it to make asynchronous HTTP requests.

The Fetch API is a modern interface that allows you to make HTTP requests to servers from web browsers.

If you have worked with XMLHttpRequest (XHR) object, the Fetch API can perform all the tasks as the XHR object does.

In addition, the Fetch API is much simpler and cleaner. It uses the [Promise](https://www.javascripttutorial.net/es6/javascript-promises/) to deliver more flexible features to make requests to servers from the web browsers.

The fetch() method is available in the global scope that instructs the web browsers to send a request to a URL.

## **Sending a Request**

The fetch() requires only one parameter which is the URL of the resource that you want to fetch:

let response = fetch(url);

Code language: JavaScript (javascript)

The fetch() method returns a Promise so you can use the then() and catch() methods to handle it:

fetch(url)

.then(response => {

*// handle the response*

})

.catch(error => {

*// handle the error*

});

Code language: JavaScript (javascript)

When the request completes, the resource is available. At this time, the promise will resolve into a Response object.

The Response object is the API wrapper for the fetched resource. The Response object has a number of useful properties and methods to inspect the response.

## **Reading the Response**

If the contents of the response are in the raw text format, you can use the text() method. The text() method returns a Promise that resolves with the complete contents of the fetched resource:

fetch('/readme.txt')

.then(response => response.text())

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

Code language: JavaScript (javascript)

In practice, you often use the [async/await](https://www.javascripttutorial.net/es-next/javascript-async-await/) with the fetch() method like this:

async function fetchText() {

let response = await fetch('/readme.txt');

let data = await response.text();

console.log(data);

}

Code language: JavaScript (javascript)

Besides the text() method, the Response object has other methods such as json(), blob(), formData() and arrayBuffer() to handle the respective type of data.

## **Handling the status codes of the Response**

The Response object provides the status code and status text via the status and statusText properties. When a request is successful, the status code is 200 and status text is OK:

async function fetchText() {

let response = await fetch('/readme.txt');

console.log(response.status); *// 200*

console.log(response.statusText); *// OK*

if (response.status === 200) {

let data = await response.text();

*// handle data*

}

}

fetchText();

Code language: JavaScript (javascript)

Output:

200

OK

Code language: JavaScript (javascript)

If the requested resource doesn’t exist, the response code is 404:

let response = await fetch('/non-existence.txt');

console.log(response.status); *// 400*

console.log(response.statusText); *// Not Found*

Code language: JavaScript (javascript)

Output:

400

Not Found

Code language: JavaScript (javascript)

If the requested URL throws a server error, the response code will be 500.

If the requested URL is redirected to the new one with the response 300-309, the status of the Response object is set to 200. In addition the redirected property is set to true.

The fetch() returns a promise that rejects when a real failure occurs such as a web browser timeout, a loss of network connection, and a CORS violation.

## **JavaScript Fetch API example**

Suppose that you have a json file that locates on the webserver with the following contents:

[{

"username": "john",

"firstName": "John",

"lastName": "Doe",

"gender": "Male",

"profileURL": "img/male.png",

"email": "john.doe@example.com"

},

{

"username": "jane",

"firstName": "Jane",

"lastName": "Doe",

"gender": "Female",

"profileURL": "img/female.png",

"email": "jane.doe@example.com"

}

]

Code language: JSON / JSON with Comments (json)

The following shows the HTML page:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>**Fetch API Demo**</title>**

**<link rel="stylesheet" href="css/style.css">**

**</head>**

**<body>**

**<div class="container"></div>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In the app.js, we’ll use the fetch() method to get the user data and render the data inside the <div> element with the class container.

First, declare the getUsers() function that fetches users.json from the server.

async function getUsers() {

let url = 'users.json';

try {

let res = await fetch(url);

return await res.json();

} catch (error) {

console.log(error);

}

}

Code language: JavaScript (javascript)

Then, develop the renderUsers() function that renders user data:

async function renderUsers() {

let users = await getUsers();

let html = '';

users.forEach(user => {

let htmlSegment = `<div class="user">

<img src="${user.profileURL}" >

<h2>${user.firstName} ${user.lastName}</h2>

<div class="email"><a href="email:${user.email}">${user.email}</a></div>

</div>`;

html += htmlSegment;

});

let container = document.querySelector('.container');

container.innerHTML = html;

}

renderUsers();

Code language: JavaScript (javascript)

Check out the [Fetch API demo](https://www.javascripttutorial.net/sample/api/fetch/index.html).

## **Summary**

* The Fetch API allows you to asynchronously request for a resource.
* Use the fetch() method to return a promise that resolves into a Response object. To get the actual data, you call one of the methods of the Response object e.g., text() or json(). These methods resolve into the actual data.
* Use the status and statusText properties of the Response object to get the status and status text of the response.
* use the catch() method or try...catch statement to handle a failure request.