Must-Know Web APIs

That Will Save Front End Developers Hours of Work

Find out more at → greatfrontend.com

LocalStorage API

The localStorage API stores key-value pairs in a user's browser with no expiration, preserving data across sessions for maintaining user preferences and state without server-side storage.

index.html × script.js ×

```
1  // Store
2  localStorage.setItem("firstname", "James");
3
4  // Retrieve
5  document.getElementById("output").innerHTML = localStorage.getItem("firstname");
6
7
8
```

Clipboard API

The Clipboard API enables web applications to read from and write to the system clipboard, primarily using the readText() and writeText() methods, which return Promises. It also includes permissions management, requiring "clipboard-read" or "clipboard-write" permissions in secure contexts

index.html × script.js ×

```
1  navigator.clipboard.writeText('Hello there!')
2    .then(() => console.log('Text copied to clipboard'));
3
4
```

Geolocation API

The Geolocation API provides user location via getCurrentPosition() and watchPosition(), using GPS, Wi-Fi, and cellular data. It requires user consent for applications like mapping and localized content.

index.html × script.js ×

```
1  navigator.geolocation.getCurrentPosition((position) => {
2    console.log(position.coords.latitude, position.coords.longitude);
3  });
4
```

History API

The History API lets developers manage browser session history with pushState(), replaceState(), and go(), enabling dynamic URL updates without reloads for a smoother single-page application experience.

```
// Load the next URL in the history list
history.forward();

// Load the previous URL in the history list
history.back();

// Load the page through its number
history.go(-2); // This will go to the previous 2nd page
history.go(2); // This will go to the next 2nd page

// Get the length of the history list
const length = history.length;

```

Fetch API

The Fetch API simplifies network requests with the fetch() method, returning a Promise that resolves to a Response object. It supports various HTTP methods, integrates with CORS, and enhances API interactions in web applications.

index.html × script.js ×

```
async function getData() {
  const url = "https://example.org/items.json";
  try {
    const response = await fetch(url);
    if (!response.ok) {
      throw new Error(`Response status: ${response.status}`);
    }

    const json = await response.json();
    console.log(json);
  } catch (error) {
    console.error(error.message);
  }
}
```

Canvas API

The Canvas API enables drawing graphics with JavaScript in the <canvas> element, supporting shapes, images, text, and animations for games and data visualization.

index.html × script.js ×

```
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");

ctx.fillStyle = "red";
ctx.fillRect(15, 15, 100, 100);

```

Notification API

The Notifications API allows web apps to send alerts outside the app, even in the background. It requires user permission and supports custom titles, icons, and interactive events.

index.html × script.js ×

```
var window: Window & typeof globalThis
1  function notifyFunc() {
2    if (!("Notification" in window)) {
3      // Check if the browser supports notifications
4      alert("Desktop notification is not supported in this browser");
5    } else if (Notification.permission === "granted") {
6      // Check whether notification permissions have already been granted;
7      // if so, create a notification
8      const notification = new Notification("Hello!");
9    } else if (Notification.permission !== "denied") {
10     // We need to ask the user for permission
11     Notification.requestPermission().then((permission) => {
12       // If the user accepts, let's create a notification
13       if (permission === "granted") {
14         const notification = new Notification("Hello!");
15       }
16     });
17   }
18 }
```

WebSocket API

The WebSocket API enables real-time, bidirectional communication over a persistent TCP connection using ws:// or wss://, ideal for chat services and live updates with low-latency interactions.

index.html × script.js ×

```
1   // Create WebSocket connection.
2   const socket = new WebSocket("ws://localhost:8080");
3
4   // Connection opened
5   socket.addEventListener("open", (event) => {
6     socket.send("Hello Server!");
7   });
8
9   // Listen for messages
10  socket.addEventListener("message", (event) => {
11    console.log("Message from server ", event.data);
12  });
```