Resurse suplimentare
pe care recomand să le parcurgi

De ce sunt importante?

- **Accesează funcționalități hardware** - Camera, microfon, senzori, GPS
- **Îmbunătățesc experiența utilizator** - Notificări, fullscreen, vibrație
- **Optimizează performanța** - Service Workers, Web Workers
- **Oferă stocare avansată** - IndexedDB, Cache API
- **Simplifică dezvoltarea** - API-uri pentru operații complexe

Categorii de Browser APIs

1. APIs pentru Locație și Senzori

2. APIs pentru Media și Hardware

3. APIs pentru Performance și Background

4. APIs pentru Stocare și Cache

5. APIs pentru Interfață și Experiență

---

1. Geolocation API - Locația utilizatorului

Ce este și când să o folosești:

Geolocation API permite accesarea locației geografice a utilizatorului. Este utilă pentru:

- Aplicații de navigație și hărți
- Servicii meteo locale
- Găsirea de magazine/servicii în apropiere
- Check-in-uri și aplicații sociale bazate pe locație

Utilizare de bază:

```javascript
// Verificarea suportului
if ('geolocation' in navigator) {
  console.log('Geolocation este suportat')
} else {
  console.log('Geolocation nu este suportat')
}

// Obținerea poziției curente
navigator.geolocation.getCurrentPosition(
  (position) => {
    const { latitude, longitude, accuracy } = position.coords
    console.log(`Lat: ${latitude}, Lng: ${longitude}`)
    console.log(`Precizie: ${accuracy} metri`)
  },
  (error) => {
    console.error('Eroare locație:', error.message)
  },
  {
    enableHighAccuracy: true,
    timeout: 10000,
    maximumAge: 600000,
  }
)
```

Versiunea modernă cu async/await:

```javascript
function getCurrentLocation() {
  return new Promise((resolve, reject) => {
    if (!navigator.geolocation) {
      reject(new Error('Geolocation nu este suportat'))
      return
    }

    navigator.geolocation.getCurrentPosition(resolve, reject, {
      enableHighAccuracy: true,
      timeout: 10000,
      maximumAge: 300000,
    })
  })
}

async function showLocation() {
  try {
    const position = await getCurrentLocation()
    const { latitude, longitude } = position.coords
    console.log(`Poziția ta: ${latitude}, ${longitude}`)

    // Afișează pe Google Maps
    const mapsUrl = `https://www.google.com/maps?q=${latitude},${longitude}`
    window.open(mapsUrl, '_blank')
  } catch (error) {
    console.error('Nu s-a putut obține locația:', error)
  }
}
```

2. Notification API - Notificări desktop

Ce este și când să o folosești:

Notification API permite afișarea de notificări în sistem, chiar și când utilizatorul nu se află pe pagina ta. Utilă pentru:

- Alerturi și reminder-e
- Mesaje noi în aplicații de chat
- Actualizări importante
- Notificări push

Utilizare:

```
// Verificarea suportului și solicitarea permisiunii
async function setupNotifications() {
  if (!('Notification' in window)) {
    console.log('Notificările nu sunt suportate')
    return false
  }

  if (Notification.permission === 'granted') {
    return true
  } else if (Notification.permission !== 'denied') {
    const permission = await Notification.requestPermission()
    return permission === 'granted'
  }

  return false
}

// Afișarea unei notificări
async function showNotification(title, options = {}) {
  const hasPermission = await setupNotifications()

  if (hasPermission) {
    const notification = new Notification(title, {
      body: options.body || 'Mesaj notificare',
      icon: options.icon || '/icon.png',
      tag: options.tag || 'default',
      ...options,
    })

    // Event listeners
    notification.onclick = () => {
      window.focus()
      notification.close()
    }

    // Auto-close după 5 secunde
    setTimeout(() => notification.close(), 5000)
  }
}

// Exemplu de utilizare
showNotification('Nou mesaj!', {
  body: 'Ai primit un mesaj de la John',
  icon: '/message-icon.png',
  tag: 'message-notification',
})
```

3. Web Speech API - Recunoaștere și sinteză vocală

Ce este și când să o folosești:

Web Speech API oferă funcționalități de speech recognition (convertirea vorbiri în text) și speech synthesis (convertirea textului în vorbire). Utilă pentru:

- Interfețe vocale și comenzi vocale
- Accesibilitate pentru utilizatori cu dizabilități
- Aplicații de dictare
- Chatbots și asistenți virtuali

Speech Recognition:

```javascript
// Verificarea suportului
const SpeechRecognition =
  window.SpeechRecognition || window.webkitSpeechRecognition

if (SpeechRecognition) {
  const recognition = new SpeechRecognition()

  recognition.continuous = true
  recognition.interimResults = true
  recognition.lang = 'ro-RO' // sau 'en-US'

  recognition.onresult = (event) => {
    let finalTranscript = ''
    let interimTranscript = ''

    for (let i = event.resultIndex; i < event.results.length; i++) {
      const transcript = event.results[i][0].transcript

      if (event.results[i].isFinal) {
        finalTranscript += transcript
      } else {
        interimTranscript += transcript
      }
    }

    console.log('Final:', finalTranscript)
    console.log('Interim:', interimTranscript)
  }

  recognition.onerror = (event) => {
    console.error('Speech recognition error:', event.error)
  }

  // Start/stop recognition
  function startListening() {
    recognition.start()
  }

  function stopListening() {
    recognition.stop()
  }
}
```

Speech Synthesis:

```
// Text-to-Speech
function speak(text, options = {}) {
  if ('speechSynthesis' in window) {
    const utterance = new SpeechSynthesisUtterance(text)

    utterance.lang = options.lang || 'ro-RO'
    utterance.rate = options.rate || 1
    utterance.pitch = options.pitch || 1
    utterance.volume = options.volume || 1

    utterance.onend = () => {
      console.log('Speech ended')
    }

    speechSynthesis.speak(utterance)
  }
}

// Exemplu de utilizare
speak('Bună ziua! Aceasta este o demonstrație de sinteză vocală.', {
  lang: 'ro-RO',
  rate: 0.8,
  pitch: 1.2,
})
```

4. Vibration API - Vibrația dispozitivului

Ce este și când să o folosești:

Vibration API permite controlul vibrației pe dispozitive mobile. Utilă pentru:

- Feedback haptic în jocuri
- Alertele și notificările
- Confirmarea acțiunilor
- Îmbunătățirea experienței tactile

Utilizare:

```javascript
// Verificarea suportului
if ('vibrate' in navigator) {
  // Vibrație simplă (200ms)
  navigator.vibrate(200)

  // Pattern de vibrație: [vibrează, pauză, vibrează, pauză, ...]
  navigator.vibrate([100, 50, 100, 50, 300])

  // Oprirea vibrației
  navigator.vibrate(0)
  // sau
  navigator.vibrate([])
}

// Funcții helper
const vibrationPatterns = {
  short: [100],
  medium: [200],
  long: [500],
  double: [100, 50, 100],
  sos: [
    100, 50, 100, 50, 100, 50, 200, 50, 200, 50, 200, 50, 100, 50, 100, 50, 100,
  ],
  notification: [200, 100, 200],
  error: [100, 50, 100, 50, 100, 50, 300],
  success: [50, 25, 50, 25, 150],
}

function vibrate(pattern) {
  if ('vibrate' in navigator && vibrationPatterns[pattern]) {
    navigator.vibrate(vibrationPatterns[pattern])
  }
}

// Utilizare
vibrate('notification') // Pentru notificări
vibrate('error') // Pentru erori
vibrate('success') // Pentru succes
```

5. Fullscreen API - Modul ecran complet

Ce este și când să o folosești:

Fullscreen API permite afișarea elementelor în modul ecran complet. Utilă pentru:

- Video players și media viewers
- Jocuri web
- Prezentări și slideshow-uri
- Aplicații immersive

Utilizare:

```
// Funcţii cross-browser pentru fullscreen
const fullscreenAPI = {
  // Verifică suportul
  isSupported() {
    return !!(
      document.fullscreenEnabled ||
      document.webkitFullscreenEnabled ||
      document.mozFullScreenEnabled ||
      document.msFullscreenEnabled
    )
  },

  // Intră în fullscreen
  enter(element = document.documentElement) {
    if (element.requestFullscreen) {
      return element.requestFullscreen()
    } else if (element.webkitRequestFullscreen) {
      return element.webkitRequestFullscreen()
    } else if (element.mozRequestFullScreen) {
      return element.mozRequestFullScreen()
    } else if (element.msRequestFullscreen) {
      return element.msRequestFullscreen()
    }
  },

  // Iese din fullscreen
  exit() {
    if (document.exitFullscreen) {
      return document.exitFullscreen()
    } else if (document.webkitExitFullscreen) {
      return document.webkitExitFullscreen()
    } else if (document.mozCancelFullScreen) {
      return document.mozCancelFullScreen()
    } else if (document.msExitFullscreen) {
      return document.msExitFullscreen()
    }
  },

  // Verifică dacă este în fullscreen
  isActive() {
    return !!(
      document.fullscreenElement ||
      document.webkitFullscreenElement ||
      document.mozFullScreenElement ||
      document.msFullscreenElement
    )
  },

  // Toggle fullscreen
  toggle(element) {
    if (this.isActive()) {
      this.exit()
    } else {
      this.enter(element)
    }
  },
}

// Exemplu de utilizare
function createFullscreenButton(targetElement) {
  const button = document.createElement('button')
  button.textContent = 'Fullscreen'

  button.addEventListener('click', () => {
    fullscreenAPI.toggle(targetElement)
  })

  // Actualizează textul butonului
  document.addEventListener('fullscreenchange', () => {
    button.textContent = fullscreenAPI.isActive()
      ? 'Exit Fullscreen'
      : 'Fullscreen'
  })

  return button
}

// Adaugă la un video element
const video = document.querySelector('video')
const fullscreenBtn = createFullscreenButton(video)
video.parentNode.appendChild(fullscreenBtn)
```

6. Intersection Observer API - Detectarea vizibilității

Ce este și când să o folosești:

Intersection Observer API detectează când elementele intră sau ies din viewport. Utilă pentru:

- Lazy loading pentru imagini
- Infinite scrolling
- Animații trigger pe scroll
- Analytics și tracking

Utilizare:

```
// Crearea unui observer simplu
const observer = new IntersectionObserver(
  (entries) => {
    entries.forEach((entry) => {
      if (entry.isIntersecting) {
        console.log('Element is visible:', entry.target)

        // Lazy load image
        if (entry.target.tagName === 'IMG' && entry.target.dataset.src) {
          entry.target.src = entry.target.dataset.src
          entry.target.removeAttribute('data-src')
          observer.unobserve(entry.target)
        }
      }
    })
  },
  {
    threshold: 0.1, // 10% din element să fie vizibil
    rootMargin: '50px', // Trigger cu 50px înainte
  }
)

// Observă toate imaginile cu data-src
document.querySelectorAll('img[data-src]').forEach((img) => {
  observer.observe(img)
})

// Observer pentru animații
const animationObserver = new IntersectionObserver(
  (entries) => {
    entries.forEach((entry) => {
      if (entry.isIntersecting) {
        entry.target.classList.add('animate-in')
      } else {
        entry.target.classList.remove('animate-in')
      }
    })
  },
  {
    threshold: 0.5,
  }
)

// Observă elementele cu clasa animate
document.querySelectorAll('.animate-on-scroll').forEach((el) => {
  animationObserver.observe(el)
})
```

7. Web Workers API - Background processing

Ce este și când să o folosești:

Web Workers permit executarea de JavaScript în background, fără a bloca UI-ul. Util pentru:

- Procesarea datelor mari
- Calcule complexe
- Manipularea imaginilor
- Background sync

Utilizare:

GENERAȚIA TECH

```
// worker.js - Fișierul worker-ului
self.onmessage = function (e) {
  const { data, action } = e.data

  switch (action) {
    case 'calculate':
      // Calcul complex
      let result = 0
      for (let i = 0; i < data.iterations; i++) {
        result += Math.sqrt(i)
      }

      // Trimite rezultatul înapoi
      self.postMessage({
        action: 'result',
        result: result,
      })
      break

    case 'processImage':
      // Procesează datele imaginii
      const imageData = data.imageData
      // ... procesare ...

      self.postMessage({
        action: 'imageProcessed',
        processedData: imageData,
      })
      break
  }
}

// main.js - Codul principal
if (window.Worker) {
  const worker = new Worker('worker.js')

  // Trimite date către worker
  worker.postMessage({
    action: 'calculate',
    data: { iterations: 1000000 },
  })

  // Primește rezultate de la worker
  worker.onmessage = function (e) {
    const { action, result } = e.data

    switch (action) {
      case 'result':
        console.log('Calculation result:', result)
        break
      case 'imageProcessed':
        console.log('Image processed')
        break
    }
  }

  worker.onerror = function (error) {
    console.error('Worker error:', error)
  }
}
```

8. Battery Status API - Informații despre baterie

Ce este și când să o folosești:

Battery Status API oferă informații despre bateria dispozitivului. Utilă pentru:

- Optimizarea consumului în aplicații
- Notificări când bateria este descărcată
- Ajustarea funcționalităților pe baza nivelului bateriei

```
// Notă: Această API este deprecated in multe browsere din motive de privacy

if ('getBattery' in navigator) {
  navigator.getBattery().then((battery) => {
    console.log('Battery level:', battery.level * 100 + '%')
    console.log('Charging:', battery.charging)
    console.log('Charging time:', battery.chargingTime)
    console.log('Discharging time:', battery.dischargingTime)

    // Event listeners pentru schimbări
    battery.addEventListener('levelchange', () => {
      console.log('Battery level changed:', battery.level * 100 + '%')
    })

    battery.addEventListener('chargingchange', () => {
      console.log('Charging status changed:', battery.charging)
    })
  })
}
```

9. Screen Orientation API - Orientarea ecranului

Ce este și când să o folosești:

Screen Orientation API permite controlul și detectarea orientării ecranului. Utilă pentru:

- Jocuri care necesită anumite orientări
- Aplicații media
- Optimizarea layout-ului

```javascript
// Verificarea orientării curente
if (screen.orientation) {
  console.log('Current orientation:', screen.orientation.type)
  console.log('Angle:', screen.orientation.angle)

  // Detectarea schimbărilor de orientare
  screen.orientation.addEventListener('change', () => {
    console.log('Orientation changed to:', screen.orientation.type)
  })

  // Forțarea unei anumite orientări (doar in fullscreen)
  async function lockOrientation(orientation) {
    try {
      await screen.orientation.lock(orientation)
      console.log('Orientation locked to:', orientation)
    } catch (error) {
      console.error('Failed to lock orientation:', error)
    }
  }

  // Deblocarea orientării
  function unlockOrientation() {
    screen.orientation.unlock()
  }

  // Exemple de utilizare
  // lockOrientation('landscape') // landscape, portrait, landscape-primary, etc.
}
```

10. Clipboard API - Clipboard modern

Ce este și când să o folosești:

Clipboard API oferă acces modern și sigur la clipboard. Utilă pentru:

- Copy/paste în aplicații
- Partajarea conținutului
- Funcționalități de editare

GENERAȚIA TECH

```
// Clipboard API modern (necesită HTTPS)
if (navigator.clipboard) {
  // Citirea din clipboard
  async function readFromClipboard() {
    try {
      const text = await navigator.clipboard.readText()
      console.log('Clipboard content:', text)
      return text
    } catch (error) {
      console.error('Failed to read clipboard:', error)
    }
  }

  // Scrierea in clipboard
  async function writeToClipboard(text) {
    try {
      await navigator.clipboard.writeText(text)
      console.log('Text copied to clipboard')
    } catch (error) {
      console.error('Failed to copy text:', error)
    }
  }

  // Exemplu de utilizare
  function createCopyButton(text) {
    const button = document.createElement('button')
    button.textContent = 'Copy'

    button.addEventListener('click', async () => {
      await writeToClipboard(text)
      button.textContent = 'Copied!'
      setTimeout(() => {
        button.textContent = 'Copy'
      }, 2000)
    })

    return button
  }
}
```