Resurse suplimentare pe care recomand să le parcurgi

localStorage vs alte metode de stocare

```
// localStorage - persistă până la ștergere manuală
localStorage.setItem('user', 'John')

// sessionStorage - se șterge la închiderea tab-ului
sessionStorage.setItem('temp', 'data')

// Cookies - trimise la server cu fiecare cerere
document.cookie = 'session=abc123; expires=...'
```

Operațiunile de bază cu localStorage

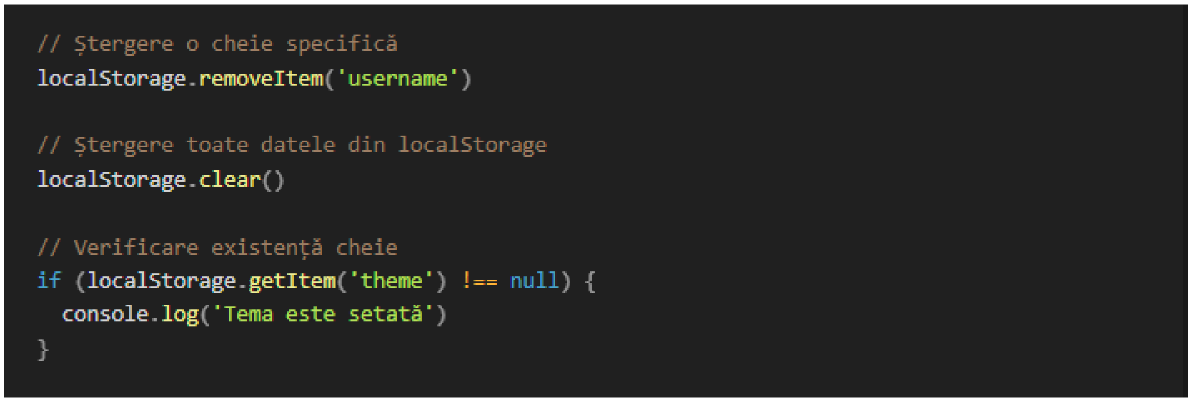
Stocarea datelor

```
// Stocare valoare simplă
localStorage.setItem('username', 'john_doe')
localStorage.setItem('theme', 'dark')
localStorage.setItem('language', 'ro')

// Stocare numere (convertite automat în string)
localStorage.setItem('userAge', 25)
localStorage.setItem('score', 1250)
```

Citirea datelor

```
// Citire valori
const username = localStorage.getItem('username')
const theme = localStorage.getItem('theme')
const language = localStorage.getItem('language')

console.log(username) // 'john_doe'
console.log(theme) // 'dark'
console.log(language) // 'ro'

// Gestionarea valorilor inexistente
const inexistent = localStorage.getItem('inexistent')
console.log(inexistent) // null
```

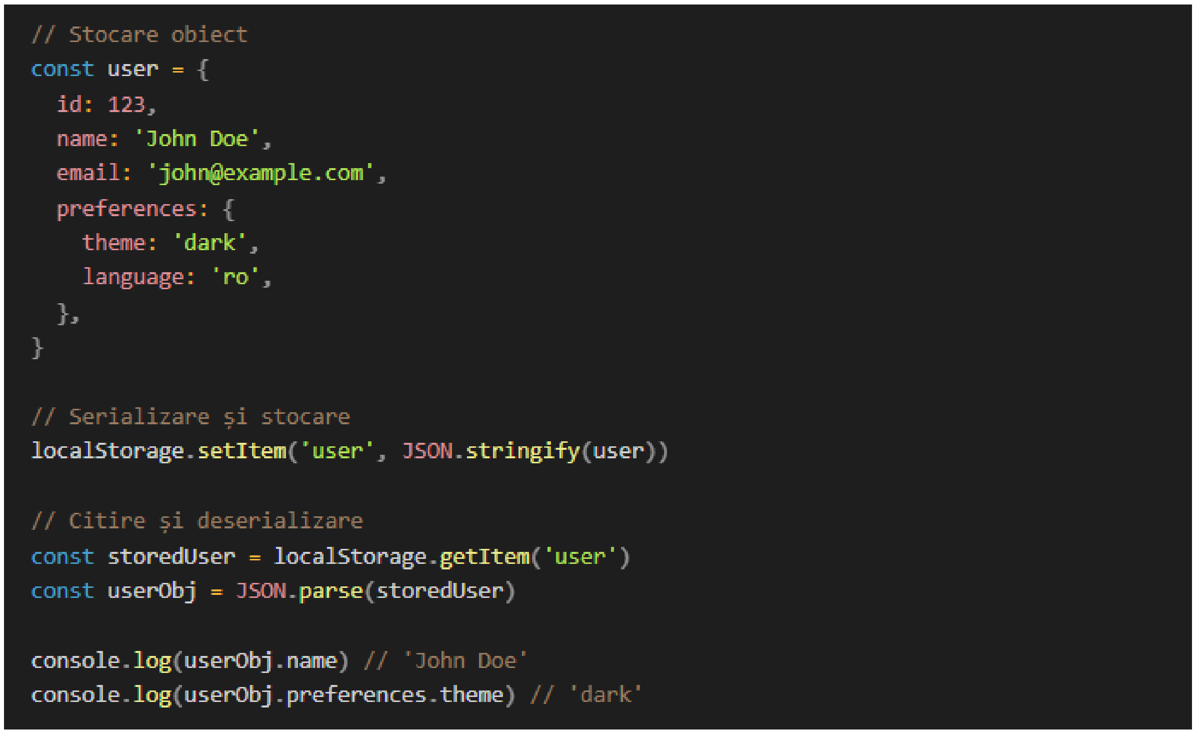
Ștergerea datelor



```
// Ștergere o cheie specifică
localStorage.removeItem('username')

// Ștergere toate datele din localStorage
localStorage.clear()

// Verificare existență cheie
if (localStorage.getItem('theme') !== null) {
  console.log('Tema este setată')
}
```

Lucrul cu obiecte și array-uri

localStorage stochează doar string-uri, deci trebuie să serializăm/deserializăm obiectele:

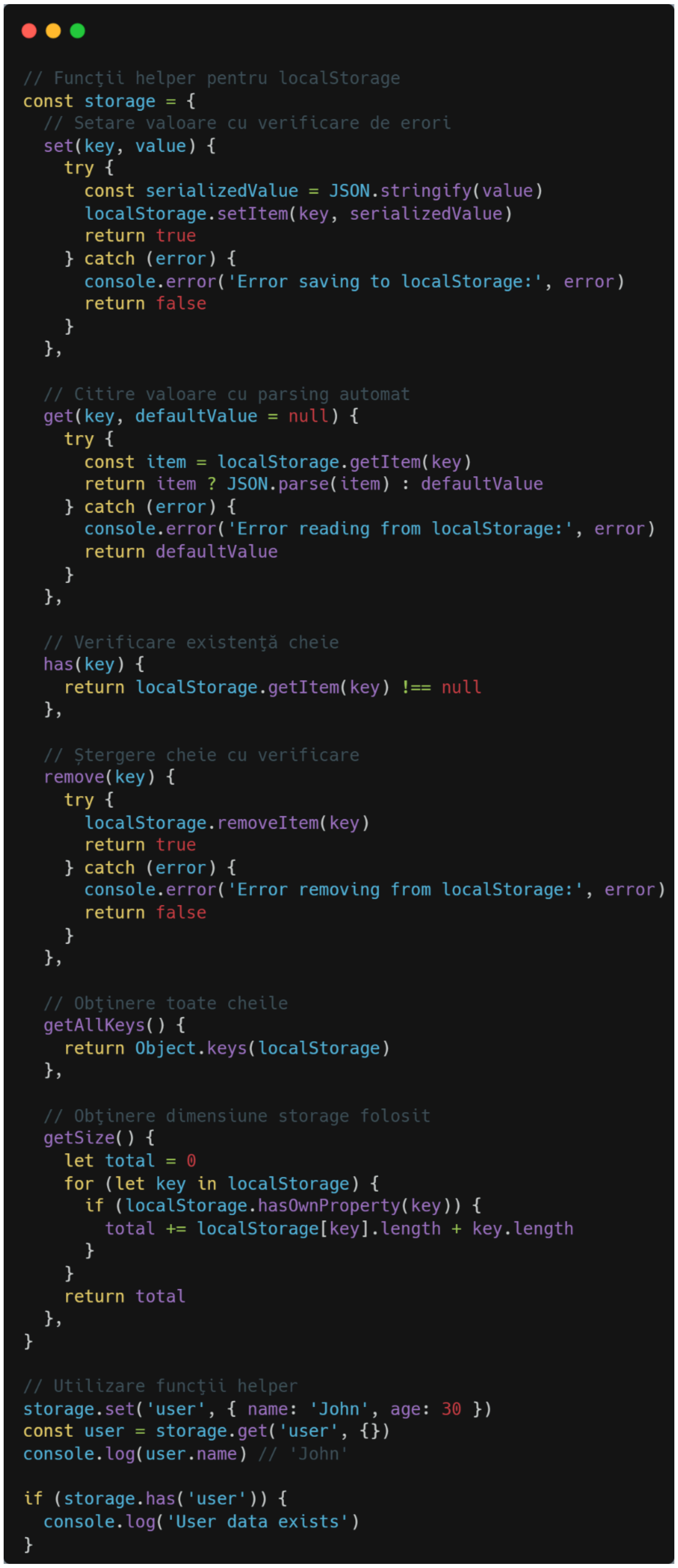


```
// Stocare obiect
const user = {
  id: 123,
  name: 'John Doe',
  email: 'john@example.com',
  preferences: {
    theme: 'dark',
    language: 'ro',
  },
}

// Serializare și stocare
localStorage.setItem('user', JSON.stringify(user))

// Citire și deserializare
const storedUser = localStorage.getItem('user')
const userObj = JSON.parse(storedUser)

console.log(userObj.name) // 'John Doe'
console.log(userObj.preferences.theme) // 'dark'
```

```javascript
// Stocare array
const tasks = [
  { id: 1, text: 'Cumpără lapte', completed: false },
  { id: 2, text: 'Finalizează proiectul', completed: true },
  { id: 3, text: 'Sună la dentist', completed: false },
]

localStorage.setItem('tasks', JSON.stringify(tasks))

// Citire și prelucrare array
const storedTasks = JSON.parse(localStorage.getItem('tasks') || '[]')
console.log(storedTasks.length) // 3
```

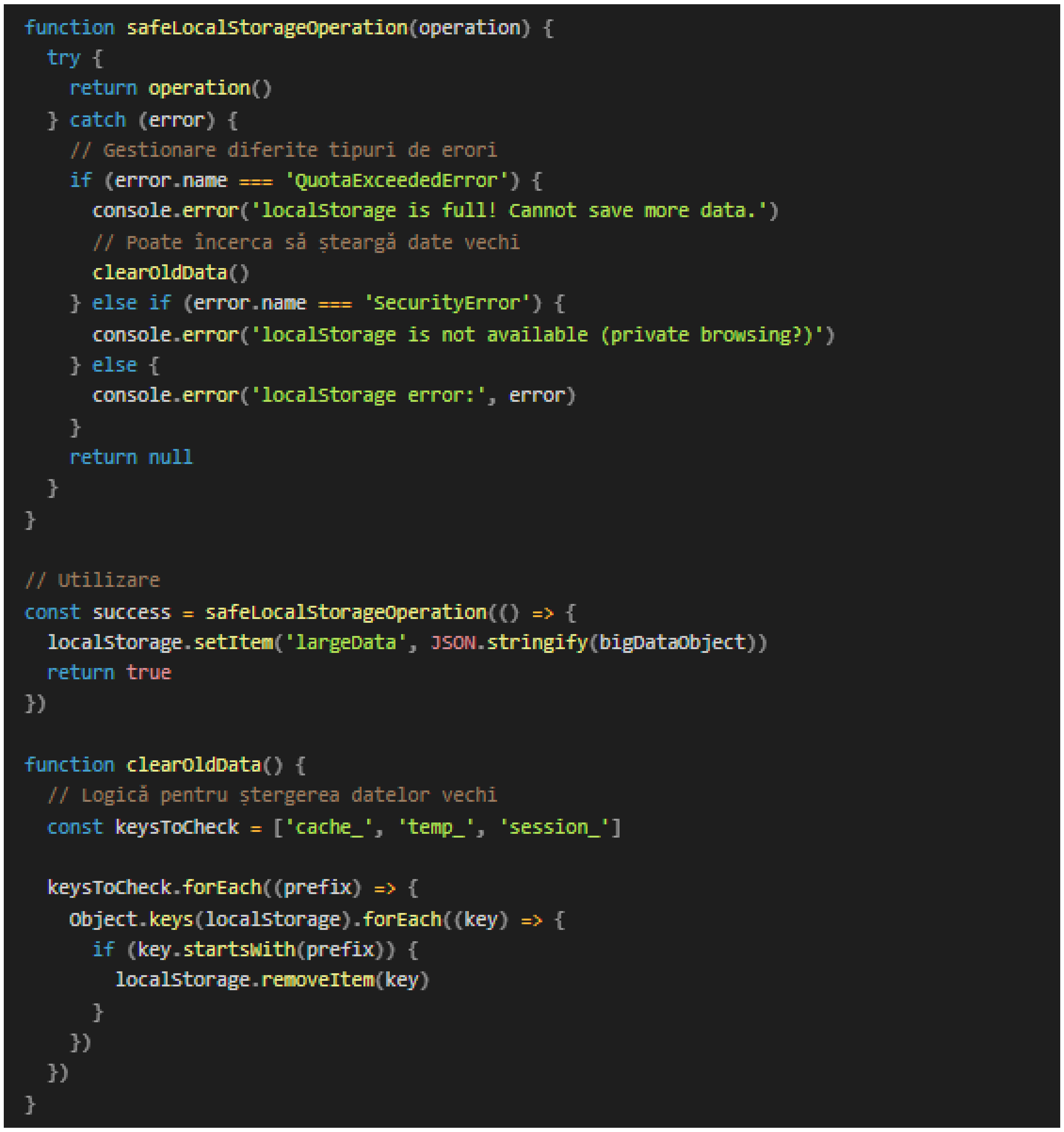
Funcții helper pentru localStorage

Să creăm funcții utile pentru a lucra mai ușor cu localStorage:



```javascript
// Funcții helper pentru localStorage
const storage = {
  // Setare valoare cu verificare de erori
  set(key, value) {
    try {
      const serializedValue = JSON.stringify(value)
      localStorage.setItem(key, serializedValue)
      return true
    } catch (error) {
      console.error('Error saving to localStorage:', error)
      return false
    }
  },

  // Citire valoare cu parsing automat
  get(key, defaultValue = null) {
    try {
      const item = localStorage.getItem(key)
      return item ? JSON.parse(item) : defaultValue
    } catch (error) {
      console.error('Error reading from localStorage:', error)
      return defaultValue
    }
  },

  // Verificare existență cheie
  has(key) {
    return localStorage.getItem(key) !== null
  },

  // Ștergere cheie cu verificare
  remove(key) {
    try {
      localStorage.removeItem(key)
      return true
    } catch (error) {
      console.error('Error removing from localStorage:', error)
      return false
    }
  },

  // Obținere toate cheile
  getAllKeys() {
    return Object.keys(localStorage)
  },

  // Obținere dimensiune storage folosit
  getSize() {
    let total = 0
    for (let key in localStorage) {
      if (localStorage.hasOwnProperty(key)) {
        total += localStorage[key].length + key.length
      }
    }
    return total
  },
}

// Utilizare funcții helper
storage.set('user', { name: 'John', age: 30 })
const user = storage.get('user', {})
console.log(user.name) // 'John'

if (storage.has('user')) {
  console.log('User data exists')
}
```

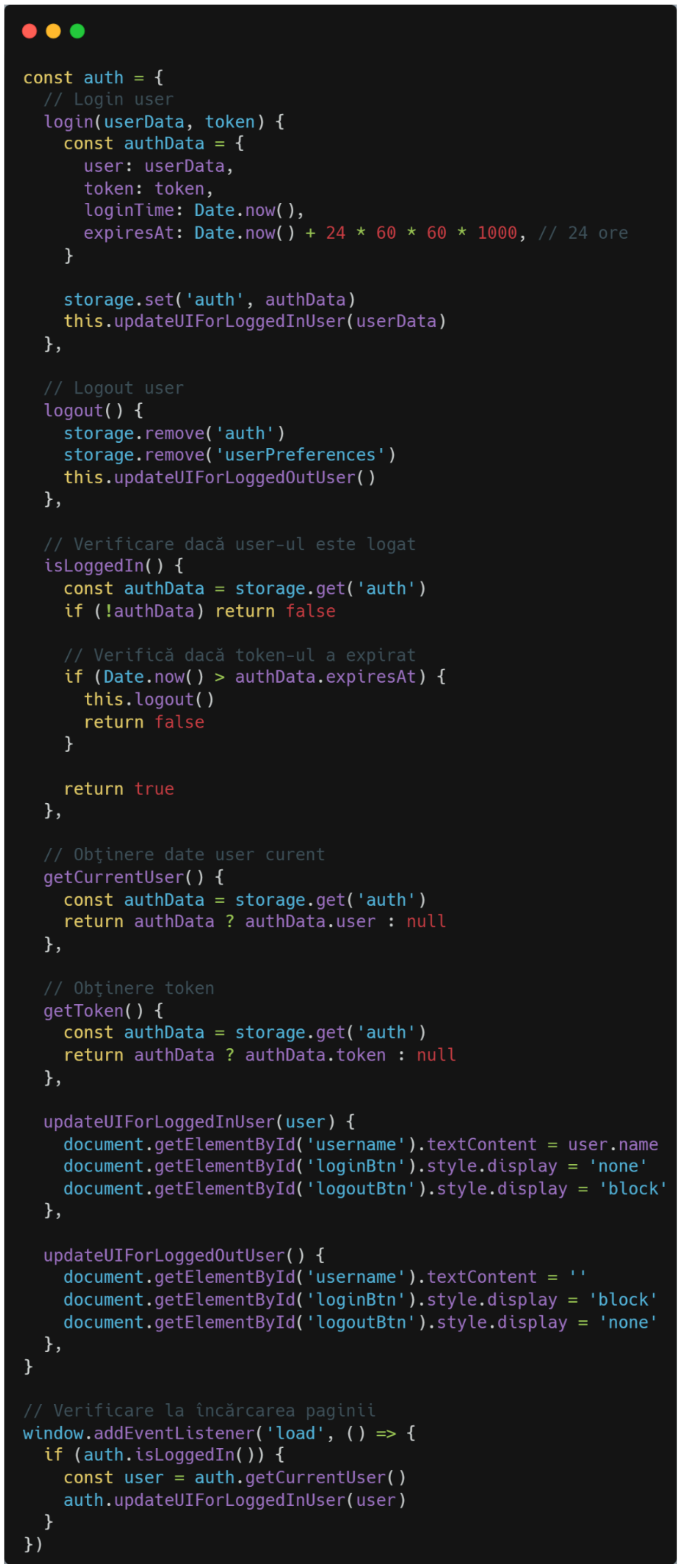
Gestionarea erorilor cu localStorage



```javascript
function safeLocalStorageOperation(operation) {
  try {
    return operation()
  } catch (error) {
    // Gestionare diferite tipuri de erori
    if (error.name === 'QuotaExceededError') {
      console.error('localStorage is full! Cannot save more data.')
      // Poate încerca să șteargă date vechi
      clearOldData()
    } else if (error.name === 'SecurityError') {
      console.error('localStorage is not available (private browsing?)')
    } else {
      console.error('localStorage error:', error)
    }
    return null
  }
}

// Utilizare
const success = safeLocalStorageOperation(() => {
  localStorage.setItem('largeData', JSON.stringify(bigDataObject))
  return true
})

function clearOldData() {
  // Logică pentru ștergerea datelor vechi
  const keysToCheck = ['cache_', 'temp_', 'session_']

  keysToCheck.forEach((prefix) => {
    Object.keys(localStorage).forEach((key) => {
      if (key.startsWith(prefix)) {
        localStorage.removeItem(key)
      }
    })
  })
}
```

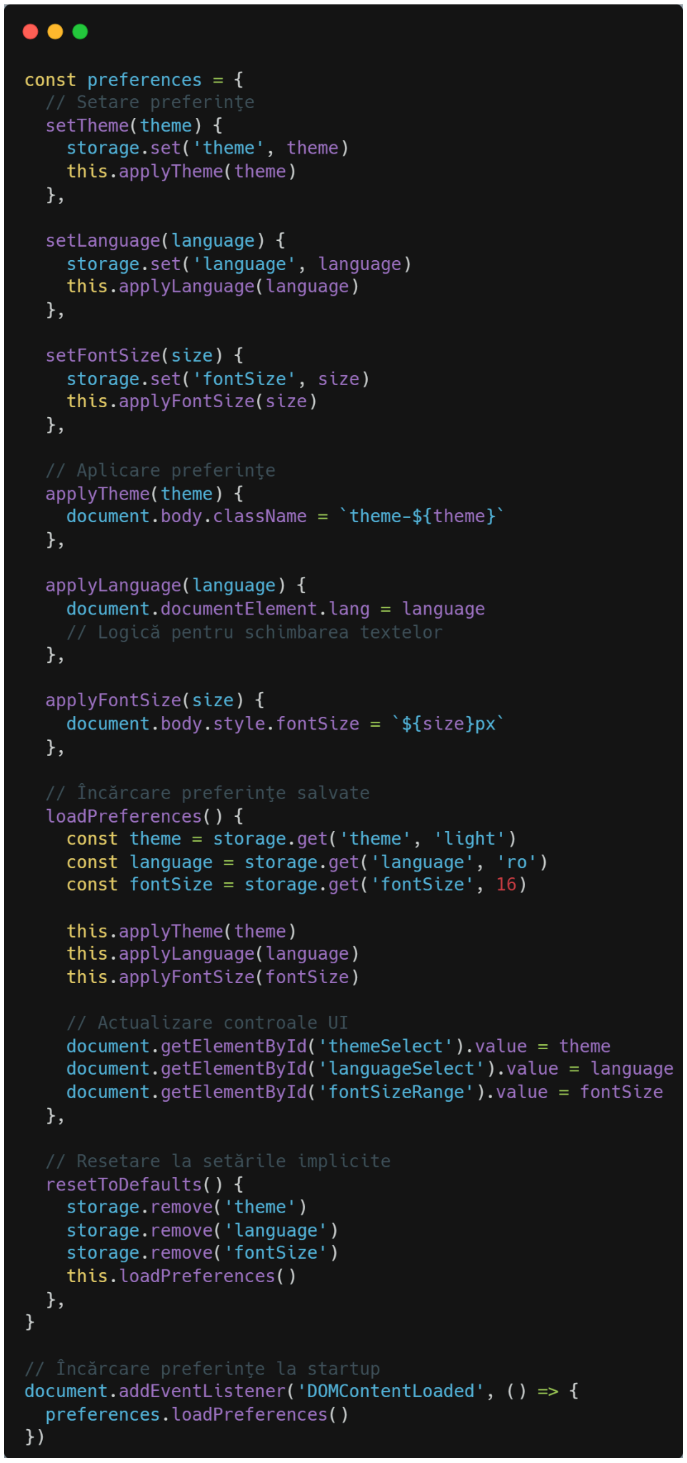
Aplicații practice cu localStorage

1. Sistem de autentificare



```javascript
const auth = {
  // Login user
  login(userData, token) {
    const authData = {
      user: userData,
      token: token,
      loginTime: Date.now(),
      expiresAt: Date.now() + 24 * 60 * 60 * 1000, // 24 ore
    }

    storage.set('auth', authData)
    this.updateUIForLoggedInUser(userData)
  },

  // Logout user
  logout() {
    storage.remove('auth')
    storage.remove('userPreferences')
    this.updateUIForLoggedOutUser()
  },

  // Verificare dacă user-ul este logat
  isLoggedIn() {
    const authData = storage.get('auth')
    if (!authData) return false

    // Verifică dacă token-ul a expirat
    if (Date.now() > authData.expiresAt) {
      this.logout()
      return false
    }

    return true
  },

  // Obţinere date user curent
  getCurrentUser() {
    const authData = storage.get('auth')
    return authData ? authData.user : null
  },

  // Obţinere token
  getToken() {
    const authData = storage.get('auth')
    return authData ? authData.token : null
  },

  updateUIForLoggedInUser(user) {
    document.getElementById('username').textContent = user.name
    document.getElementById('loginBtn').style.display = 'none'
    document.getElementById('logoutBtn').style.display = 'block'
  },

  updateUIForLoggedOutUser() {
    document.getElementById('username').textContent = ''
    document.getElementById('loginBtn').style.display = 'block'
    document.getElementById('logoutBtn').style.display = 'none'
  },
}

// Verificare la încărcarea paginii
window.addEventListener('load', () => {
  if (auth.isLoggedIn()) {
    const user = auth.getCurrentUser()
    auth.updateUIForLoggedInUser(user)
  }
})
```

2. Salvarea preferințelor utilizatorului

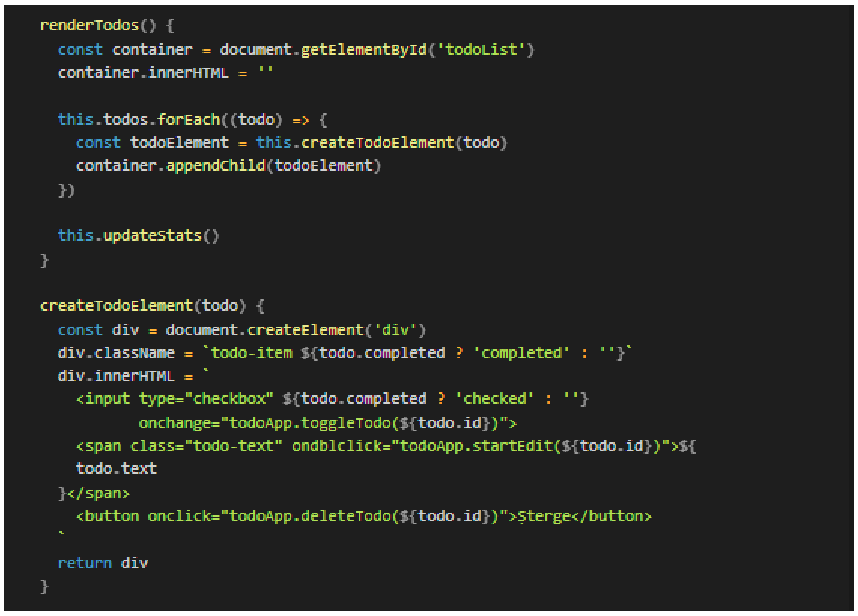


```javascript
const preferences = {
  // Setare preferinţe
  setTheme(theme) {
    storage.set('theme', theme)
    this.applyTheme(theme)
  },

  setLanguage(language) {
    storage.set('language', language)
    this.applyLanguage(language)
  },

  setFontSize(size) {
    storage.set('fontSize', size)
    this.applyFontSize(size)
  },

  // Aplicare preferinţe
  applyTheme(theme) {
    document.body.className = `theme-${theme}`
  },

  applyLanguage(language) {
    document.documentElement.lang = language
    // Logică pentru schimbarea textelor
  },

  applyFontSize(size) {
    document.body.style.fontSize = `${size}px`
  },

  // Încărcare preferinţe salvate
  loadPreferences() {
    const theme = storage.get('theme', 'light')
    const language = storage.get('language', 'ro')
    const fontSize = storage.get('fontSize', 16)

    this.applyTheme(theme)
    this.applyLanguage(language)
    this.applyFontSize(fontSize)

    // Actualizare controale UI
    document.getElementById('themeSelect').value = theme
    document.getElementById('languageSelect').value = language
    document.getElementById('fontSizeRange').value = fontSize
  },

  // Resetare la setările implicite
  resetToDefaults() {
    storage.remove('theme')
    storage.remove('language')
    storage.remove('fontSize')
    this.loadPreferences()
  },
}

// Încărcare preferinţe la startup
document.addEventListener('DOMContentLoaded', () => {
  preferences.loadPreferences()
})
```

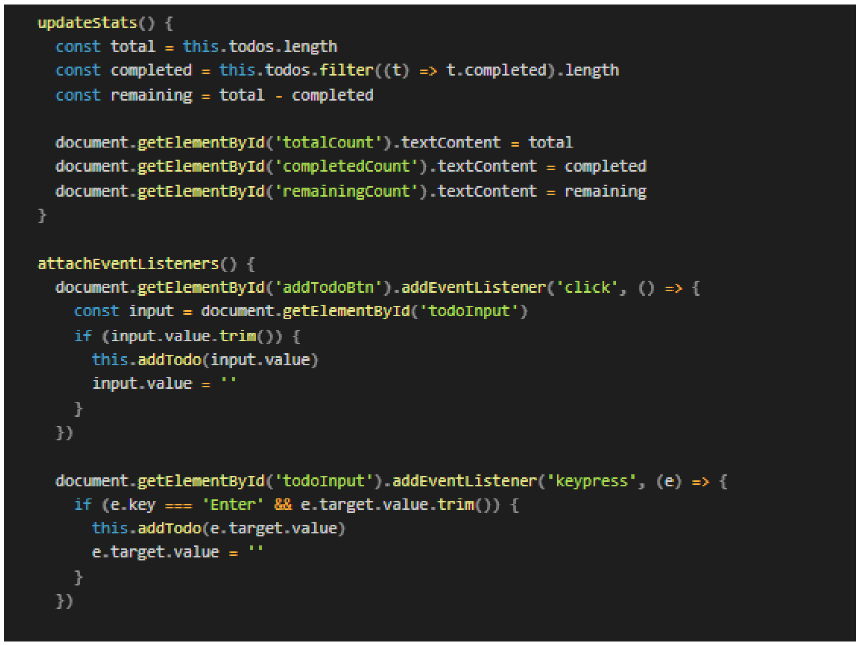
3. Lista de TODO cu localStorage

```
class TodoApp {
  constructor() {
    this.todos = storage.get('todos', [])
    this.nextId = storage.get('nextTodoId', 1)
    this.init()
  }

  init() {
    this.renderTodos()
    this.attachEventListeners()
  }

  addTodo(text) {
    const todo = {
      id: this.nextId++,
      text: text.trim(),
      completed: false,
      createdAt: Date.now(),
    }

    this.todos.push(todo)
    this.saveTodos()
    this.renderTodos()
  }
```

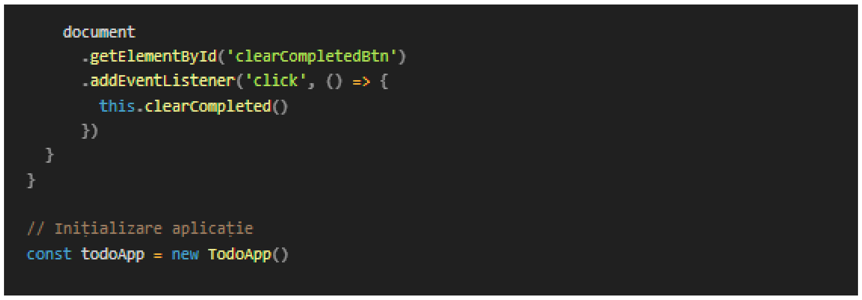
```
toggleTodo(id) {
  const todo = this.todos.find((t) => t.id === id)
  if (todo) {
    todo.completed = !todo.completed
    this.saveTodos()
    this.renderTodos()
  }
}

deleteTodo(id) {
  this.todos = this.todos.filter((t) => t.id !== id)
  this.saveTodos()
  this.renderTodos()
}

editTodo(id, newText) {
  const todo = this.todos.find((t) => t.id === id)
  if (todo && newText.trim()) {
    todo.text = newText.trim()
    this.saveTodos()
    this.renderTodos()
  }
}

clearCompleted() {
  this.todos = this.todos.filter((t) => !t.completed)
  this.saveTodos()
  this.renderTodos()
}

saveTodos() {
  storage.set('todos', this.todos)
  storage.set('nextTodoId', this.nextId)
}
```



```
  renderTodos() {
    const container = document.getElementById('todoList')
    container.innerHTML = ''

    this.todos.forEach((todo) => {
      const todoElement = this.createTodoElement(todo)
      container.appendChild(todoElement)
    })

    this.updateStats()
  }

  createTodoElement(todo) {
    const div = document.createElement('div')
    div.className = `todo-item ${todo.completed ? 'completed' : ''}`
    div.innerHTML = `
      <input type="checkbox" ${todo.completed ? 'checked' : ''}
             onchange="todoApp.toggleTodo(${todo.id})">
      <span class="todo-text" ondblclick="todoApp.startEdit(${todo.id})">${
      todo.text
    }</span>
      <button onclick="todoApp.deleteTodo(${todo.id})">Șterge</button>
    `
    return div
  }
```



```
  updateStats() {
    const total = this.todos.length
    const completed = this.todos.filter((t) => t.completed).length
    const remaining = total - completed

    document.getElementById('totalCount').textContent = total
    document.getElementById('completedCount').textContent = completed
    document.getElementById('remainingCount').textContent = remaining
  }

  attachEventListeners() {
    document.getElementById('addTodoBtn').addEventListener('click', () => {
      const input = document.getElementById('todoInput')
      if (input.value.trim()) {
        this.addTodo(input.value)
        input.value = ''
      }
    })

    document.getElementById('todoInput').addEventListener('keypress', (e) => {
      if (e.key === 'Enter' && e.target.value.trim()) {
        this.addTodo(e.target.value)
        e.target.value = ''
      }
    })
```



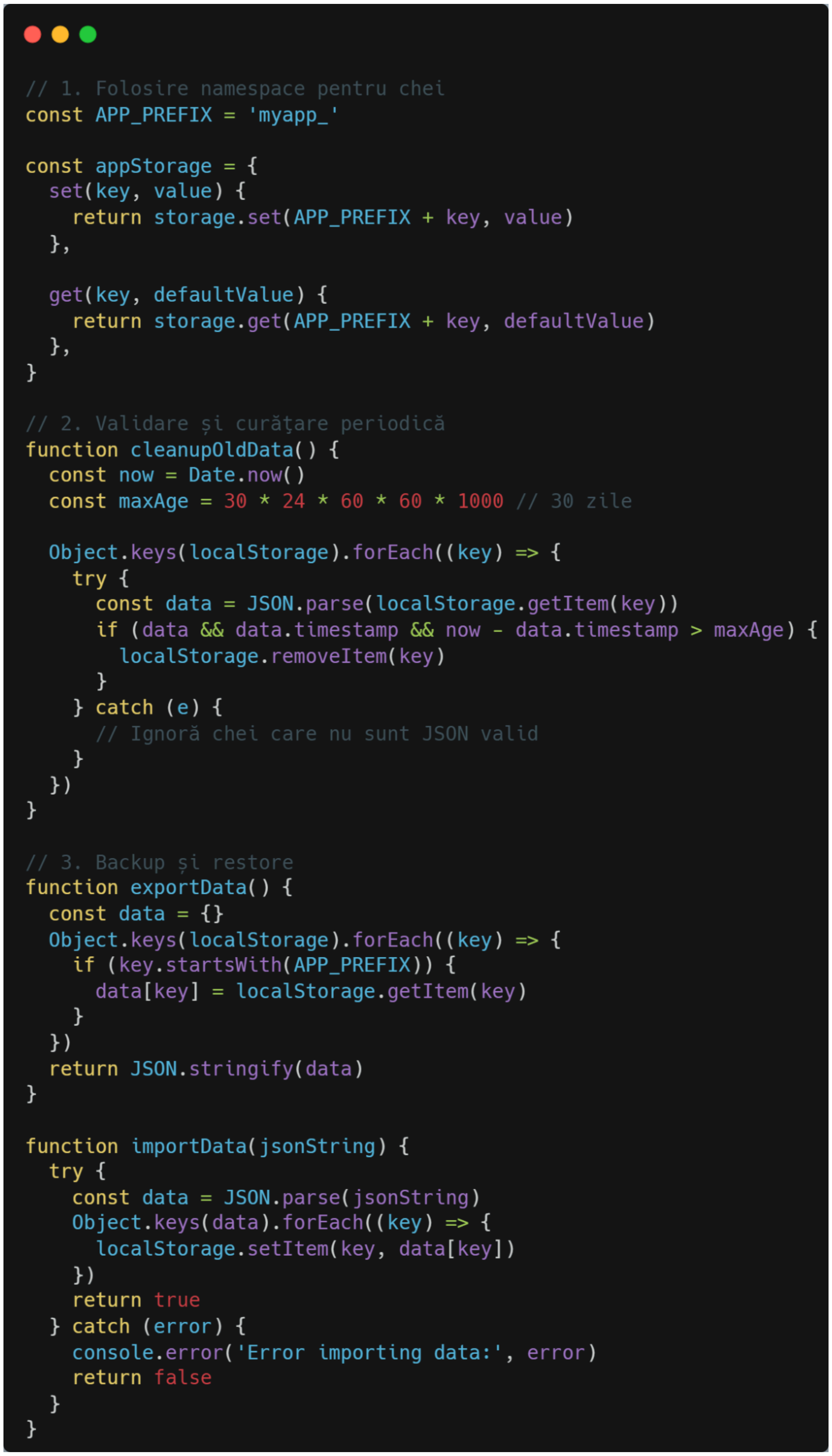
```
    document
      .getElementById('clearCompletedBtn')
      .addEventListener('click', () => {
        this.clearCompleted()
      })
  }
}

// Inițializare aplicație
const todoApp = new TodoApp()
```

Limitări și best practices

Limitări localStorage

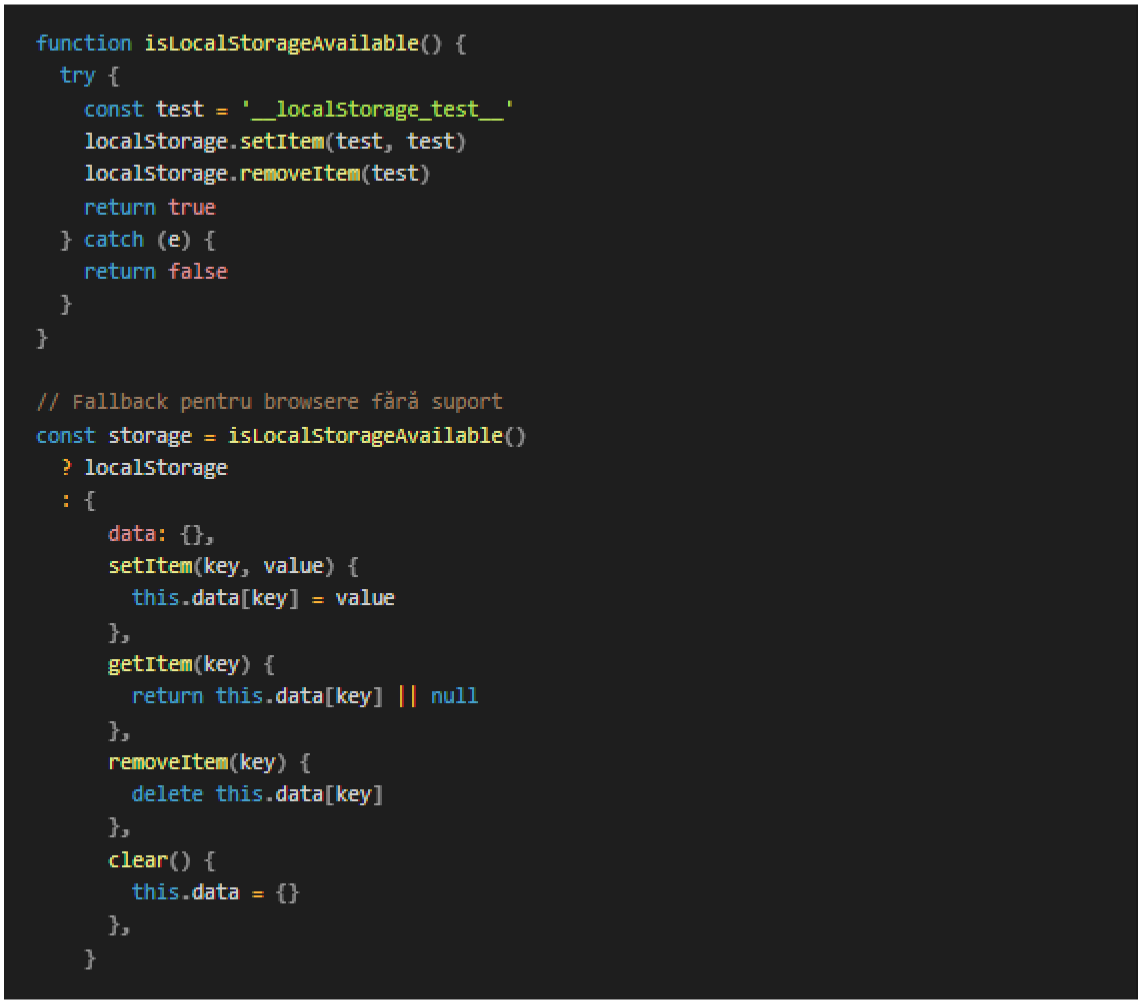
```javascript
// 1. Capacitate limitată (5-10MB)
function checkStorageQuota() {
  let total = 0
  for (let key in localStorage) {
    if (localStorage.hasOwnProperty(key)) {
      total += localStorage[key].length + key.length
    }
  }

  const totalKB = Math.round(total / 1024)
  console.log(`localStorage folosit: ${totalKB} KB`)

  if (totalKB > 4096) {
    // ~4MB
    console.warn('localStorage se apropie de limită!')
  }
}

// 2. Doar string-uri (nu obiecte native)
// ❌ Greșit
localStorage.setItem('date', new Date())
localStorage.setItem('func', function () {})

// ✅ Corect
localStorage.setItem('date', new Date().toISOString())
localStorage.setItem('config', JSON.stringify({ setting: 'value' }))

// 3. Sincron - poate bloca UI-ul
// Pentru date mari, consideră alternatives
```

Best Practices



```javascript
// 1. Folosire namespace pentru chei
const APP_PREFIX = 'myapp_'

const appStorage = {
  set(key, value) {
    return storage.set(APP_PREFIX + key, value)
  },

  get(key, defaultValue) {
    return storage.get(APP_PREFIX + key, defaultValue)
  },
}

// 2. Validare și curățare periodică
function cleanupOldData() {
  const now = Date.now()
  const maxAge = 30 * 24 * 60 * 60 * 1000 // 30 zile

  Object.keys(localStorage).forEach((key) => {
    try {
      const data = JSON.parse(localStorage.getItem(key))
      if (data && data.timestamp && now - data.timestamp > maxAge) {
        localStorage.removeItem(key)
      }
    } catch (e) {
      // Ignoră chei care nu sunt JSON valid
    }
  })
}

// 3. Backup și restore
function exportData() {
  const data = {}
  Object.keys(localStorage).forEach((key) => {
    if (key.startsWith(APP_PREFIX)) {
      data[key] = localStorage.getItem(key)
    }
  })
  return JSON.stringify(data)
}

function importData(jsonString) {
  try {
    const data = JSON.parse(jsonString)
    Object.keys(data).forEach((key) => {
      localStorage.setItem(key, data[key])
    })
    return true
  } catch (error) {
    console.error('Error importing data:', error)
    return false
  }
}
```

Detectarea suportului pentru localStorage



```javascript
function isLocalStorageAvailable() {
  try {
    const test = '__localStorage_test__'
    localStorage.setItem(test, test)
    localStorage.removeItem(test)
    return true
  } catch (e) {
    return false
  }
}

// Fallback pentru browsere fără suport
const storage = isLocalStorageAvailable()
  ? localStorage
  : {
      data: {},
      setItem(key, value) {
        this.data[key] = value
      },
      getItem(key) {
        return this.data[key] || null
      },
      removeItem(key) {
        delete this.data[key]
      },
      clear() {
        this.data = {}
      },
    }
```