Ljubljana.tech

endava

Enhancing web applications with persistent browser storage

JURE ROTAR

FRONTEND ENGINEER @ ENDAVA

endava

Link to slides

https://github.com/jurerotar/presentations/blob/main/enhancing-web-applications-with-persistent-browser-storage.pdf

Why store data in the browser?

Reduced server load

Faster access to resources

Customization & personalization

Offline access & privacy

Prototyping and development velocity

Browser storage types

Web storage API - **local storage** & **session storage**

Cache storage

IndexedDB

File system - **File system access** & **OPFS**

Shared storage

Local/session storage

Very simple API

Works with key-value pairs

local storage - no expiration date

session storage - expires at the end of the session

```
// Store a value
localStorage.setItem('username', 'JohnDoe');

// Retrieve a value
const username = localStorage.getItem('username');
console.log(username); // Outputs: 'JohnDoe'

// Remove a value
localStorage.removeItem('username');

// Clear all values
localStorage.clear();
```

Common use cases:

- Saving user preferences
- Remembering form data

Web storage API downsides

Non-async, blocking API

Only supports string values

Fast by itself, but slows down with stringification

No indexing support

5 MB limit

Cache storage

Request-response pair storage

No set expiration date (!)

50 - 500 MB size limit

Supports multiple cache instances

Allow very granular cache control

```
// Open a cache named 'my-cache'
const cache = await caches.open('my-cache');

// Add an item to the cache
await cache.add('/assets/image.jpg');

// Retrieve an item from the cache
const response = await cache.match('/assets/image.jpg');

// Delete the cache
await caches.delete('my-cache');
```

Common use cases:

- Static asset caching
- API response caching
- Versioned assets caching
- Offline functionality support

endava

Cache storage downsides

Requires manual cache control

Users may end up with stale content, if cache is incorrectly managed

It’s important to handle versioning correctly, otherwise your app might break

IndexedDB

Structured data storage

Very expansive API

No set expiration (except Safari)

Large storage size (500 MB - 2 GB)

Transactions & indexing support

```
// Open or create a database named 'my-database' with version 1
const db = await indexedDB.open('my-database', 1);

// Create a transaction and object store
const transaction = db.transaction(['storeName'], 'readwrite');
const store = transaction.objectStore('storeName');

// Add data to the object store
await store.add({ id: 1, name: 'John Doe', email: 'john@example.com' });

// Retrieve data from the object store
const data = await store.get(1);
```

Common use cases:

- Media storage (audio, video)
- File storage
- Memory-efficient data filtering

IndexedDB downsides

Complex native API (fixed with wrappers like dexie.js)

Performance issues with increasing number of transactions

Noticeable slower read/write performance than alternatives

Tons of Safari-specific bugs:

- IndexedDB operations hang without progress or error
- Backing file on disk (WAL file) keeps growing
- QuotaExceededError on first use
- There’s a couple more, but I’m out of space 😬

Origin private file system

Supports text & binary data

Size limited to 10-80% of disk space

No set expiration date

Creates sandboxed files on user FS

Supports data streaming

```ts
export const writeFile = async (name: string, data: string) => {
    // Get the root directory for storage
    const root = await navigator.storage.getDirectory();
    // Get a file handle (create if it doesn't exist)
    const file = await root.getFileHandle(name, { create: true });
    // Create a synchronous access handle
    const syncHandle = await file.createSyncAccessHandle();
    // Write the data to the file
    syncHandle.write((new TextEncoder()).encode(data));
    // Close the access handle
    syncHandle.close();
};
```

Common use cases:

- Video, audio & image editing
- Browser game development
- Browser-based databases
- Document storage

OPFS downsides

No support for streaming in Safari

Safari only supports sync read & write operations:

- Both are render-blocking
- You will want to offload these to a web worker
- This introduces additional complexity

Data may get cleared, if device is low on available storage

File system access

Direct access to user files

Real-time changes on user files

Security focused

Supports modern file operations

```
// Request permission to access the file system
const [handle] = await window.showOpenFilePicker();

// Get a file from the user's file system
const file = await handle.getFile();

// Read the contents of the file
const text = await file.text();

// Display the file contents
console.log('File contents:', text);
```

Enables these use cases:

- Text editors
- Photo albums
- Spreadsheet editor

File system access downsides

Implemented only in Chromium based-browsers

Rejected by both Mozilla & WebKit for its potential security issues

Shared storage

Cross-origin data sharing:

- Allows feature flag management across different origins
- Allows your preferences to persist through different origins without abusing cookies

Removes the need to duplicate data / requests across different origins

Includes strong privacy & security controls to prevent tracking

Shared storage downsides

Expires 30 days after last write

Implemented only in Chromium based-browsers

Rejected by both Mozilla & WebKit

Feature comparison

Storage --- Feature	Local storage	Session storage	Cache storage	IndexedDB	File system access	OPFS
Supports async	No	No	Yes	Yes	Yes	Yes
Availability	window	window	window, web & service workers	window, web & service workers	window	window, web workers
Size limit	5 MB / origin	5 MB / origin	50 MB - 500 MB / origin	500 MB - 2 GB / origin	Disk space	10-80% of disk space / origin
Expiration	Never	End of session	Never	Never	Never	Never

Local/session storage

Window API: localStorage

Chrome	Edge *	Safari	Firefox	Opera	IE *
		3.1-3.2	2-3	10.1	6-7
4-124	12-124	4-17.4	3.5-125	11.5-108	8-10
125	125	17.5	126	109	11
126-128		17.6-TP	127-129		

Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
	3.2-17.4	4-23		12-12.1		2.1-4.4.4				2.5
125	17.5	24	all	80	15.5	125	126	14.9	13.52	3.1
	17.6									

Cache storage

CacheStorage API

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠ *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-39				10-26												
1 40-42	12-15	3.1-11	2-40	1 27-29			3.2-11.2									
43-128	16-127	11.1-17.6	41-129	30-110	6-10		11.3-17.6	4-24		12-12.1		2.1-4.4.4				2.5
129	128	18.0	130	111	11	128	18.0	25	all	80	15.5	128	127	14.9	13.52	3.1
130-132		18.1-TP	131-133				18.1									

IndexedDB

indexedDB API

Chrome	Edge *	Safari	Firefox	Opera	IE *
			2-9		
4-23		3.1-7.1	10-15	10-12.1	6-9
24-124	12-124	8-17.4	16-125	15-108	10
125	125	17.5	126	109	11
126-128		17.6-TP	127-129		

Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
	3.2-7.1					2.1-4.3				
	8-17.4	4-23		12-12.1		4.4-4.4.4				2.5
125	17.5	24	all	80	15.5	125	126	14.9	13.52	3.1
	17.6									

File system access

File System Access API

Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-73	12-18			10-60												
1 74-85	1 79-85			1 62-71												
2 86-104	2 86-104			2 72-90												
105-128	105-127	3.1-17.6	2-129	91-110	6-10		3.2-17.6	4-24		12-12.1		2.1-4.4.4				2.5
129	128	18.0	130	111	11	128	18.0	25	all	80	15.5	128	127	14.9	13.52	3.1
130-132		18.1-TP	131-133				18.1									

Origin private file system

FileSystemFileHandle API

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠ *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-85	12-85	3.1-15.1	2-110	10-71			3.2-15.1	4-13.0								
86-124	86-124	15.2-17.4	111-125	72-108	6-10		15.2-17.4	14.0-23		12-12.1		2.1-4.4.4				2.5
125	125	17.5	126	109	11	125	17.5	24	all	80	15.5	125	126	14.9	13.52	3.1
126-128		17.6-TP	127-129				17.6									

FileSystemFileHandle API: createWritable

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠ *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-85	12-85		2-110	10-71				4-13.0								
86-124	86-124	3.1-17.4	111-125	72-108	6-10		3.2-17.4	14.0-23		12-12.1		2.1-4.4.4				2.5
125	125	17.5	126	109	11	125	17.5	24	all	80	15.5	125	126	14.9	13.52	3.1
126-128		17.6-TP	127-129				17.6									

Shared storage

SharedStorage API

Chrome	Edge *	Safari	Firefox	Opera	IE *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-116	12-116			10-102				4-23								
117-128	117-127	3.1-17.6	2-129	103-110	6-10		3.2-17.6	24		12-12.1		2.1-4.4.4				2.5
129	128	18.0	130	111	11	128	18.0	25	all	80	15.5	128	127	14.9	13.52	3.1
130-132		18.1-TP	131-133				18.1									

Testing

Browser storage APIs are often not part of test environments

Check your test environment specifications

Make sure you include required polyfills

Commonly used polyfills:

`vitest-localstorage-mock / jest-localstorage-mock`

`fake-indexeddb`

`vitest-opfs-mock`

Interactive whiteboard

Client’s users requested a interactive whiteboard feature:

- Needs to include image/document upload
- Needs to include a collaboration mode
- Must not include external solutions, data can’t leave the app
- Needs to be implemented into their existing app

Problems

- We had no data to estimate how used this feature would be:
  - This made it hard to push the feature higher on the priority scale
- Backend & UI/UX teams had very low capacity:
  - Availability was estimated in 2-3 months
  - Backend work estimated at 3-4 months
- Project would most certainly get delayed even more

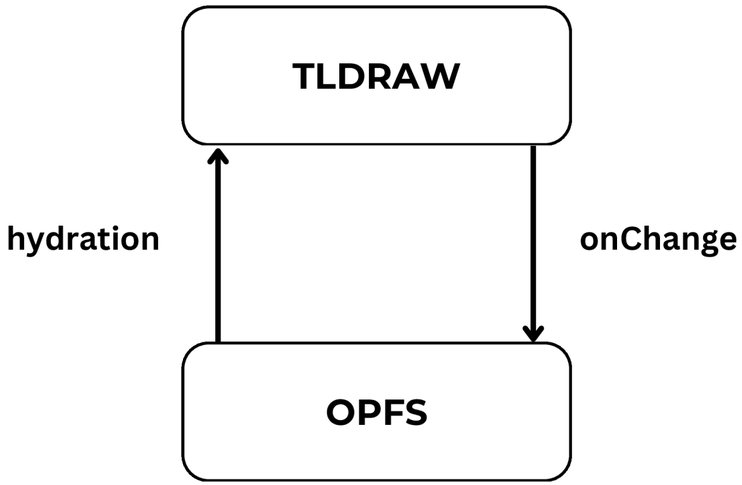
(Potential?) solution

- Frontend team had available capacity to create a PoC
- We estimated a limited PoC (without collaboration mode) would take just a week to implement
- We figured best way to gather required data is by letting real users use it

Prototyping

We used tldraw library with OPFS as persistence solution:

- Hydrate tldraw state from OPFS on app start
- Write new state to OPFS on every change



Results

- PoC got in users' hands in less than a week
- Test period lasted for 2 weeks
- Number of daily users grew from 15% of eligible users on day 1, to 70% of eligible users at the end of the test period
- Data gathered from initial users lead to a shift in priorities
- Initial collaboration mode was implemented in less then 2 months
- OPFS is still used in offline mode

Browser storage is powerful. Use it!

- Enhances performance
- Improves user experience
- Enables rapid prototyping
- Enhances privacy