TodoMVC indexedDB workload for Speedometer

Attention - this doc is public and shared with the world!

Author: Luis Pardo

Last update: Aug 11, 2025

Link to hosted prototype: Speedometer 3

Link to prototype's code: Speedometer/resources/todomvc/vanilla-examples/javascript-wc-indexeddb at lpardosixtos/WCIndexeddb · lpardosixtosMs/Speedometer

Summary

IndexedDB has been flagged as one of the areas that are currently missing representation in Speedometer. This document presents a simple extension of todoMVC for the next version of Speedometer.

Proposal

Link to hosted prototype: Speedometer 3

Link to prototype's PR: Add TodoMVC with indexedDB storage by lpardosixtosMs · Pull Request #523 · WebKit/Speedometer

We propose to incorporate some indexedDB interaction on top of the todoMVC workload. We are changing the structure of the to-do list from an "endless" list to a paged list with 10 items per page; this will allow us to do chunk reads from the indexedDB in the third step. This proposal intentionally avoids *awaiting* on every single store or load as I think it is unrealistic and feels like a micro-benchmark.

Current prototype's database schema: A simple db with 5 Indices.

```
const store = db.createObjectStore(this.storeName, { keyPath: 'itemNumber' });
store.createIndex('id', 'id', { unique: true });
store.createIndex('title', 'title', { unique: false });
store.createIndex('completed', 'completed', { unique: false });
store.createIndex('priority', 'priority', { unique: false });
```

Step 1. Add 100 items.

The first step is almost the same as the original. It'll add 100 items to the list but will also perform 100 additions to the indexedDB.

Measured time

The measured time must cover all the interaction + layout object. Specific to indexed DB we have 2 options:

Option 1: Stop the timers once all the "success" events have been processed in the main thread. This option would allow us to optimize for better write performance (but do we really want this?).

Option 2 (preferred): Stop the timers when the layout is completed. This won't explicitly measure any indexedDB related time, but it will measure the impact that the user might see by having the page performing indexedDB additions in the background.

Step 2. Complete 100 items.

There will be 10 iterations, on each iteration all the items in the page will be marked completed and the user will move to the next page. When moving to the next page, the items in the current page will be deleted from memory because they'll be retrieved from the indexedDB later. Every time an item is completed, the corresponding entry will be updated in the DB.

Measured time

Same 2 options as with step 1.

Step 3. Delete 100 items.

There will be 10 iterations, on each iteration all the items in the page will be deleted and the user will move to the previous page. To move to the previous page, the previous 10 items must be loaded from the database, we await for this load to happen.

Measured time

As the DB reads are awaited, the DB read time is already included in the measured time. We have the same 2 options as with the other steps to decide when to stop the clock.

Implementation details

Libraries

The current prototype already has an implementation using the vanilla indexedDB API, but from feedback from our internal experts, most developers use other wrappers like Dexie.js - Minimalistic IndexedDB Wrapper because of the complexity and poor

ergonomics of the vanilla API. (**Open question: Is the group ok with the choice of dexie.js)**.

Access patterns

The current proposal is only testing data loading by chunks, not individual values. This is something that we could modify. Additionally, the current implementation is using cursors to load the data, this may not be the right approach.

Stressing variables

While the proposed workload is very simple, we could add variables to make it more realistic. For example:

- DB size (artificially increased in a non-measured step).
- Multiple db/stores operations. We could create multiple dbs and stores and update or query them at the same time to simulate possible contention.

Data types

The proposed scenario is just loading and storing js arrays with text and number values. I don't think storing blobs, images or videos can fit into the todoMVC story, but perhaps we could propose some modifications to the NewSites workloads or our in-progress responsive design workload.