**Replacing Local Storage with IndexedDB**

When you build a [PWA](https://en.wikipedia.org/wiki/Progressive_web_applications), you often have to store data locally. The easiest way is to use [LocalStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage). Data persists even after ending sessions and it does easy synchronous read-write operation. Although you need to convert a JSON object to string because LocalStorage only supports string, the programming overhead is minimum.

For the basic data manipulation, you can just do this. It is not asynchronous, so the code runs after each LocalStorage call will reflect the change to the store. Easy. No worrries.

<script src="https://gist.github.com/mydatahack/1b9fc59f0445d71029e16db270e101f6.js"></script>

This is as far as it goes with LocalStorage. The simple read-write operation. Instead of having smart logic to update data, it just does bulk insert and replace the old data with new one. LocalStorage also has the limit of 10MB storage space.

PWAs sometimes require more complex data operation with real NoSQL capability. Storing strings with key-value pairs may not be good enough. App may needs to store other data types like blobs. Then, it is time to use IndexedDb.

**Introducing IndexedDb**

[IndexedDb](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API) is a client-side storage that works like NoSQL that has transactional capability. It has a rich set of APIs to do transactional operations. Google also recommends IndexedDb for the go-to storage for PWA.

In reality, using IndexedDb is not easy because it works only asynchronously and API is quirky and not very friendly to handle it as it is not Promise based.

If you have worked with backend application with database calls, the last thing you want is that database operation is not getting done synchronously. For example, when you update data, retrieve it, and manipulate it in three consecutive lines, you really want it to run synchronously as you don’t want to data retrieval and manipulation after data update completes.

IndexedDb doesn’t work that way. In JavaScript, promise is a nice way to turn an asynchronous execution into a synchronous mode. Unfortunately, IndexedDb APIs are not promise friendly. Instead, you need to listen to an event and fire an action. Of course, we can wrap the whole thing in promise (which I will show you how later), but that seems like a lot of effort.

Although a bit of effort is required to get over the learning curve, IndexedDb is super powerful. It will opens up the whole new world of real database operation on the browser. I am actually stoked that modern browsers have such a rich database out of the box.

So, I am hoping that this post will help you to be familiar with IndexedDb’s quirky APIs.

Let’s start with basics.

**Basic database operation**

First, we need to create a database, then create a store (that is like a table). In the store, we want to insert data. Then we want to read it. This sounds simple. But, it gets complex because IndexedDb works asynchronously in every step with emitting events.

To learn this quirky API, I needed to write this in a scripting language way. No function, just execute the whole thing in a script tag in an HTML. Then, when I got this working, I sort of understood how IndexedDb works.

One more caveat is that the file has to run in a server. You cannot just open HTML file in a browser. You can use gulp or webpack to run it in a local server to test the code below.

You could create an auto increment key and do indexing (as in the commented out section). But, this is mainly for a simple operation to replace the local storage logic at the start of this post.

These APIs emit events and you need to catch them to do further operation. Arrrg…

<script src="https://gist.github.com/mydatahack/26d063f30963c60326bf072fe88b4afe.js"></script>

**Make it Promise Friendly**

The script above is not usable in the real application. The solution for this is to make it promise friendly. Then, we can do async/await pattern that is more pleasant for database operation.

In fact, [google documentation](https://developers.google.com/web/ilt/pwa/working-with-indexeddb) recommends to use [promise-friendly library](https://github.com/jakearchibald/idb) to interact with IndexedDb. Yeah, this library is pretty nice. But, let’s try to DIY this time.

1. **InitialiseDB**

This will initialise the database and store and add an array of data. As you can see, I wrapped the whole thing in promise.

<script src="https://gist.github.com/mydatahack/cddd36193c632af1af3a29c3206aab44.js"></script>

1. **Fetching Data**

Opening database is asynchronous as well as reading data. So, I opted to create two functions rather than nesting promises. The first one returns the store, then use the store object to read data in the second function.

<script src="https://gist.github.com/mydatahack/1f06501b4210a398c9a2b138195a078b.js"></script>

1. **Inserting Data**

Once the target object store is available, we can to data insertion operation.

<script src="https://gist.github.com/mydatahack/7448fbc752f39e4f6b3f3c692d257601.js"></script>

1. **Clearing database**

Yeah, you can clear database this way. I actually observed delete database operation gets blocked a lot. You can catch blocked event with onblock on the request object. Even if it gets blocked, deletion seems to happen eventually. So, I didn’t bother to add onblock event in this function.

<script src="https://gist.github.com/mydatahack/cfd77ae26ed9725e0415a223cf65d12a.js"></script>

**Putting it all together**

Ok, let’s write an array of key-value pair data and read it from IndexedDb when the page is first loaded.

<script src="https://gist.github.com/mydatahack/11e30440e842b1647a904287213c2338.js"></script>

**Final thoughts**

The 100 plus lines of code to do simple operation with IndexedDb in this example can be done in a few lines of code in LocalStorage. But, that is not the point. IndexedDb can do so much more and if you are creating PWA apps that requires database transaction data operation while offline, you should totally go for IndexedDb. In fact, APIs for IndexedDb have been changing a bit. It used to have synchronous operations, but deprecated it. I think API usability will improve over the time.

If your app only requires to read and write simple key-value pair string data, IndexedDb might be an over-engineered solution and you can totally go with LocalStorage.

Well, let’s see how browser storage technology evolves. I have no doubt this blog will become obsolete very soon.