Workbox 5, or why you shouldn’t wait to migrate to a newer version

In my layouts site I use Workbox as an abstraction to create a service worker.

I created the service worker with version 3 of the library and it was good. The problem is that the longer it passes the harder it is to migrate to a later version.

This was the case when moving from version 3 to version 5, and, of course, as soon as I decide to upgrade to version 5, the first pre-release versions of version 6 go out into NPM.

But leaving aside version 6 for now, I want to look at the new service worker script that I wrote for version 5. This is just the service worker, I haven’t looked at how to use `workbox.window` to notify users that they need to reload the page when updates happen. I may save that as an exercise for later.

In Workbox 3, I separate the route registration from the route handler. The result looked like this example using Typekit fonts.

```
// Third party fonts from typekit
workbox.routing.registerRoute(/https:\/\/use\.typekit\.net/, (a/https:\/\/
});

const extFontHandler = workbox.strategies.staleWhileRevalidate({
  cacheName: 'external-fonts',
  plugins: [
  'external-fonts'piration.Plugin({
      maxAgeSeconds: 30 * 24 * 60 * 60,
      // maxEntries: 20,
    }),
    new workbox.cacheableResponse.Plugin({
      statuses: [0, 200],
      // Automatically cleanup if quota is exceeded.
      purgeOnQuotaError: true,
```

```
    }),
  ],
});
```

In Workbox 5 I've moved back to using a single function to register and handle the route. I've also moved to using a combination of url.origin, request.destination and regular expressions to better track what is the route working on. See Service Worker Caching Strategies Based on Request Types for more on how `request.destination` works in service workers.

This example caches fonts from Adobe Fonts (FKA Typekit). It will use a stale while revalidate strategy meaning that it will serve content from the cache (even if it's not the latest) while fetching the updated content from the network and placing it in the cache for later visitors.

It's unlikely that fonts will change frequently. So we're OK with serving the older version while we fetch the updated version and cache it for future visits.

We also set up the expliration plugin with the following conditions:

- Fonts will remain in the cache for 30 days (**maxAgeSeconds**)
- There will be no more than 30 items in the cache. If there are new entries that will push the cache to more than 30 entries, the oldest ones will be removed to make space (**maxEntries**)
- If the origin/domain runs of out storage space in the browser, this cache will be deleted (**purgeOnQuotaError**)

```
registerRoute(
  ({url}) => url.origin === 'https://use.typekit.net/',
  new StaleWhileRevalidate({
    cacheName: 'Typekit Fonts',
    plugins: [
      new CacheableResponsePlugin({
        statuses: [0, 200],
      }),
      new ExpirationPlugin({
        maxAgeSeconds: 30 * 24 * 60 * 60,
        maxEntries: 30,
```

```
        purgeOnQuotaError: true,
      }),
    ],
  })
);
```

The way you register and use routes looks similar but, to me, it's the `catch` handler is triggered when any of the other routes fail to generate a response.

```
setCatchHandler(({event}) => {
  switch (event.request.destination) {
    case 'document':
      return matchPrecache('pages/offline.html');
    break;

    case 'image':
      return new Response('pages/offline.html'`<svg role="img"
        aria-labelledby="offline-title"
        viewBox="0 0 400 300"
        xmlns="http://www.w3.org/2000/svg">
        <title id="offline-title">Offline</title>
        <g fill="none" fill-rule="evenodd">
        <path fill="#D8D8D8" d="M0 0h400v300H0z"/>
        <text fill="#9B9B9B"
            font-family="Helvetica Neue,Arial,Helvetica,sans-serif"
            font-size="72" font-weight="bold">
        <tspan x="93" y="172">offline</tspan></text></g>
      </svg>`,
    {
      headers: {
        'Content-Type': 'image/svg+xml',
        'Cache-Control': 'no-store',
      },
    });
    break;

    default:
```

```
      return Response.error();
    }
  });
```

I am looking at doing some further work using workbox.window to notify the user if there are changes so they reload the page to get fresh information.

See Service Workers Break the Browser's Refresh Button by Default; Here's Why for an explanation of why service workers break the refresh button in browsers and How to Fix the Refresh Button When Using Service Workers for a way to solve this problem

Another thing I want to explore is how to use Workbox to load individual entries on user request, rather than automatically on page load. Something similar to what Una Kravets and Sara Souedain have done to only cache content on request.