Service Worker tools update

I haven't looked at Service Workers in a while and wondered how much the SW libraries have changed since I last looked at them.

SW-precache and SW-toolbox made creating Service Workers with dynamic caching much easier but there were two separate libraries and SW-precache required a separate file with all the SW-toolbox libraries routes in it. This made it very error prone to edit and update.

Workbox.js is the evolution of Google's Service Worker Libraries. It consolidates all Service Worker build steps into one task (Gulp, Webpack and NPM Script versions available) and abstracts a lot of the writing and configuration behind the scenes so developers don't need to see the process, only the result.

Workbox is a Node package so I always install it as a development dependency with the command below

```
npm install workbox-build --save-dev
```

At the top of the `gulpfile.js` file place the following constant declaration to bring workbox-build into scope of the file.

```
const wbBuild = require('workbox-build');
```

Copy the task below to your gulpfile. Note that this task uses ES6 arrow functions and promises so it'll only work on newer versions of Node (5.x and newer).

```
gulp.task('bundle-sw', () => {
  return wbBuild.generateSW({
   globDirectory: './_site/', './_site/'// 1t: './_site/sw.js', // 2
    s'./_site/sw.js'// 2*\/*.{html,js,css}'], // 3
    globIgnores: ['sw.js'], // 4
    skipWaiting: tru'sw.js'// 4
```

```
    skipWaiting: true, // optional
    clientsClaim: true, // optionalole.log('Service worker generated.');
  })
  .catch((err) => {
    console.log('[ERROR] ', err);
  });
})
'Service worker generated.'
```

The tasks tells `workbox-build`:

1. Where to search for content
2. Where to write the resulting Service Worker
3. What files to add to the Service Worker (all HTML, CSS and Javascript files)
4. What files to ignore. In this example we don't want to cache the service worker itself as caching would defeat the purpose

If it succeeds then we log a success message to console and if we fail we log the error to console as well.

The last bit that I need to research is how to add routes to handle the cases that are not handled in the task as currently written. SW-toolbox provided multiple ways to configure caching strategies for multiple items in the same application.

There is a portion of workbox, `workbox-routing` that handles the routing portion of the Service Worker but I haven't quite figured out how to integrate it to the gulp file using `workbox-build` to generate the manifest and the precaching of assets. I've asked the developers who created the tools and they've pointed to examples where the routes are configured manually... There has to be a way to do it programmatically.

Thanks to Jeff Posnick for pointing me to a solution to integrate `workbox-routing` and `workbox-build` on the same Service Worker.

The solution is a two-step process. We first write our Service Worker as shown below. We pass an empty array as the parameter to `workboxSW.precache` because we'll populate this from the `service-worker` task in Gulp.

The routes we define in the service worker will not change or will not change to frequently so doing it this way makes sure we get the best of both worlds.

The service worker belows mirrors the `toolbox-scripts` file that we created using the old `sw-toolbox` library.

```
// Alternatively, use your own local copy of workbox-sw.prod.vX.Y.Z.js
importScripts('https://unpkg.com/workbox-sw@0.0.1');

const workboxSW = new goog.SWL'https://unpkg.com/workbox-sw@0.0.1'our dev
// A// Pass in an empty array for our dev environment service worker.ulp t
// current the precache manifest.
workboxSW.precache([]);

// All navigation requests should be routed to the App Shell.
// since we're not using app shell it's commented out
// workboxSW.router.registerNavigationRoute('/');

// Use a cache first strategy for files from googleapis.com
workboxSW.'re not using app shell it'// current the precache manifest.rkbo
    cacheName: 'googleapis',
    cacheExpiration: {
      // Expire after 3 days (expressed in seconds)
      maxAgeSeconds: 3 * 24 * 60 * 60
    }
  })
);

// Use a cache-first strategy for the images
workboxSW.router.registerRoute(
  new RegExp('/.(?:png|gif|jpg)$/'),
  workboxSW.strategies.cacheFirst({
    cacheName: 'images',
    cacheExpiration: {
      // maximum 50 entries
      maxEntries: 50,
      // Expire after 30 days (expressed in seconds)
      maxAgeSeconds: 30 * 24 * 60 * 60
    },
    // The images are returned as opaque responses, with a status of 0.
    // Normally these wouldn't be cached; here we opt-in to caching them.
```

```
    // If the image returns a satus 200 we cache it too
    cacheableResponse:'googleapis'[0, 200]}
  })
);

// Match all// Expire after 3 days (expressed in seconds)SW.router.registe
  new RegExp('/.html$|.htm$/'),
  workboxSW.strategies.cacheFirst({
    cacheName: 'content',
    cacheExpiration: {
      maxAgeSeconds: 1 * 24 * 60 * 60
    }
  })
);

// For video we use a network only strategy. We don't want to clog
// the cache with large video files
workboxSW.router.registerRoute(
  new RegExp('/.(?:youtube|vimeo).com$/'),
  workboxSW.strategies.networkOnly()
);

// The default route uses a cache first strategy
workboxSW.router.registerRoute('/*',
  workboxSW.strategies'/.html$|.htm$/'// For video we use a network only s
  workboxSW.strategies.cacheFirst()
);
```

The modified service worker takes a different approach than what we saw before using `workbox-build`. Innstead of building the manifest directly, it injects the list of files in the manifest into the service worker. Remember that we put an empty array on the precache section of the service worker. This is the task that will populate the empty array with the files we need to precache.

And the best part is that, if we missed anythhing, the files will be cached at run time. Not optimal but we will not loose any content.

```
gulp.task('service-worker', () => {
  return workboxBuild.injectManifest({
    swSrc: 'src/service-worker.js',
 'src/service-worker.js'ce-worker.js',
    globDirectory: '_site',
    staticFileGlobs: [
      'rev/js'_site'/**/      'rev/styles/*.css''rev/styles/*.css'/*.css'
      'images/**/*'
    ]
  });
});
```

In static or content heavy sites we may want to further constraint the values for step 3 as the default value may cache too many files and make the initial cache and subsequent retrievals take longer than we'd like.