How Caching for the Frontend Works

There are several different levels of file caching for the Frontend service. They are as follows.

Note: This does not cover how the Frontend caches application data that is retrieved from the Backend service.

Service Worker Caching

Once the Service Worker (SW) is installed and activated, it takes over all direct caching responsibilities.

All file requests (i.e. everything that isn't an AJAX request) goes through the SW, which then checks its precache for the file.

All files that are generated at build time are put into this precache, so the app can basically function completely without having to hit the Express server.

Cache Updates

The SW cache is updated whenever the user updates the app.

The user will be prompted to update the app whenever the there are new updates and they have either refreshed the page or used the "Check for Updates" settings item.

How these updates work is that the `/service-worker.js` file (which defines how the SW works) has a manifest of files that were generated during the build step, along with revision numbers (i.e. hashes) for all files that aren't hash-named (e.g. `index.html`).

'Checking for updates' just means that the `/service-worker.js` file is pulled from the Express server and check for any changes to the precache manifest. If there are changes (e.g. if the revision on `index.html` changes), then there is a new update and the user will be prompted to apply this update.

Because we want users to be able to manually check for updates whenever they want, it is important that the `service-worker.js` file is *not* cached by Nginx, and is instead served fresh from the Express server every time. This way, users can get updates as soon as they are published.

- Note: There is an argument to be made that we *could* cache the `service-worker.js` file in Nginx, if at least for a shorter period of time. This would just mean that there is a delay when rolling out updates and when users can get them. Which is theoretically fine, but while I'm still in heavy development like I currently am, I'd rather be able to check for immediate updates.

Cache Busting

The SW cache can be busted by just logging out (and refreshing the page). This will cause the SW to be unregistered. Then, when the user logs back in, the SW will be registered again, but it'll have all of the latest files.

HTTP Caching

All static files (except `index.html` and `service-worker.js`) are cached using the browser's HTTP cache.

This is controlled by setting the `Cache-Control` header on the file response to `public, max-age=86400`. Note: 86400 seconds = 1 day.

This header is set by the `express.static` middleware that is used in the Frontend Express server.

This level of caching is used whenever the SW isn't registered (i.e. while the user is logged out).

It is important that `index.html` doesn't get cached here so that the user can refresh the page and get the latest JS/CSS when they are logged out.

It is important that `service-worker.js` doesn't get cached here for the reasons stated above in "Service Worker Caching".

Cache Updates

As the `max-age` property would suggest, items in this cache are updated once they go beyond their max age (currently 1 day).

Cache Busting

This cache can be busted/bypassed by hard reloading the page while Dev Tools are open (in Chrome), or while "Disable Cache" is enabled in Dev Tools.

Nginx Ingress Caching

The Nginx ingress that is used as the entry point into the Kubernetes cluster also has its own caching.

The purposes of this caching level is to lower the number of requests that make it to the Express server, which is definitely slower than whatever Nginx can serve out of its cache.

Items are populated into the Nginx cache based on their `Cache-Control` headers. If the header is set with `public, max-age=whatever`, then the file will be cached and served from Nginx. If the header is set to `no-cache`, then Nginx will not cache the file and all requests for the file will be served from their origin (in this case, the Frontend Express server).

Note: Nginx doesn't use the `max-age` property to determine how long something is stored in the cache. Instead, this is controlled by the `proxy_cache_valid` property in the `ingress.yaml` template. It is currently set to 24 hours, coincidentally the same as the current `max-age` property.

Cache Updates

Files in the Nginx cache will be marked as stale and re-populated once the `proxy_cache_valid` time elapses (currently, as stated above, it is 24 hours).

Cache Busting

The Nginx cache can be manually cleared by following the steps in How to Clear Nginx Cache.

Express Server Caching

The final level of 'caching' is handled by the Express server that serves the actual Frontend service.

However, this is a different form of caching than all of the above. All the Express server caches is the 'rendered' `index.html` file.

As you might recall, the `index.html` file has a `script` tag that specifies all of the configuration for how the Frontend client can access the Backend service (e.g. host, port, etc).

These configuration values use placeholder values until the `index.html` file is 'rendered' by the Express server using its environment variable values. Additionally, the hash for the `script` tag is calculated so that the CSP headers can be set correctly. The result of this rendering process is just a string that is stored (i.e. cached) in a variable on the server.

This way, the Express server only has to render the file once, at boot up, rather than for every request.

This is the only 'caching' that the Express server does. Everything else is just served from disk.

Cache Updates/Busting

This 'rendered' file cache will be updated/busted whenever the server is restarted. Most notably, this happens whenever a new version of the app is deployed by the CI/CD pipeline.

As such, since the values being injected (Backend configuration values) are static per-branch, having this level of cache updating/busting is fine, since it never *needs* to change once deployed, but it *can* change easily with just a re-deploy.