



A Deep Dive into Caching with Service Workers

Kat Fairbanks

Kat Fairbanks

Full stack developer
Cincinnati community organizer

kat@craftsmanship.dev
@derkatzenbar

GENERATE





**I can do that
with Javascript!**



What are service workers?

- A special kind of Web Worker that runs on a separate thread from the main page's Javascript, sometimes even when the page isn't even open in the browser!
- Can be used for push notifications on the web.
- Can act as a proxy between a web page and the network. The service worker can decide to respond to network requests directly using cached data.
 - Used to prefetch and cache frequently used data in the background to improve application performance.
 - Used to allow access to web pages offline.
- Only available in secure browsing contexts (https or localhost).

Examples of websites using service workers

- Offline Support
 - Excalidraw — <https://excalidraw.com/>
 - Regex101 — <https://regex101.com/>
 - YouTube — Allows you to download videos while online, for viewing offline later (paid feature, but great for getting tech talks in on long flights)
 - Spotify — Has an offline shell, but does not have the download functionality available
 - Google Docs, Slides — Can edit documents offline
- Caching
 - <https://www.kroger.com/> — caches lots of images for long-term use

Registering a service worker

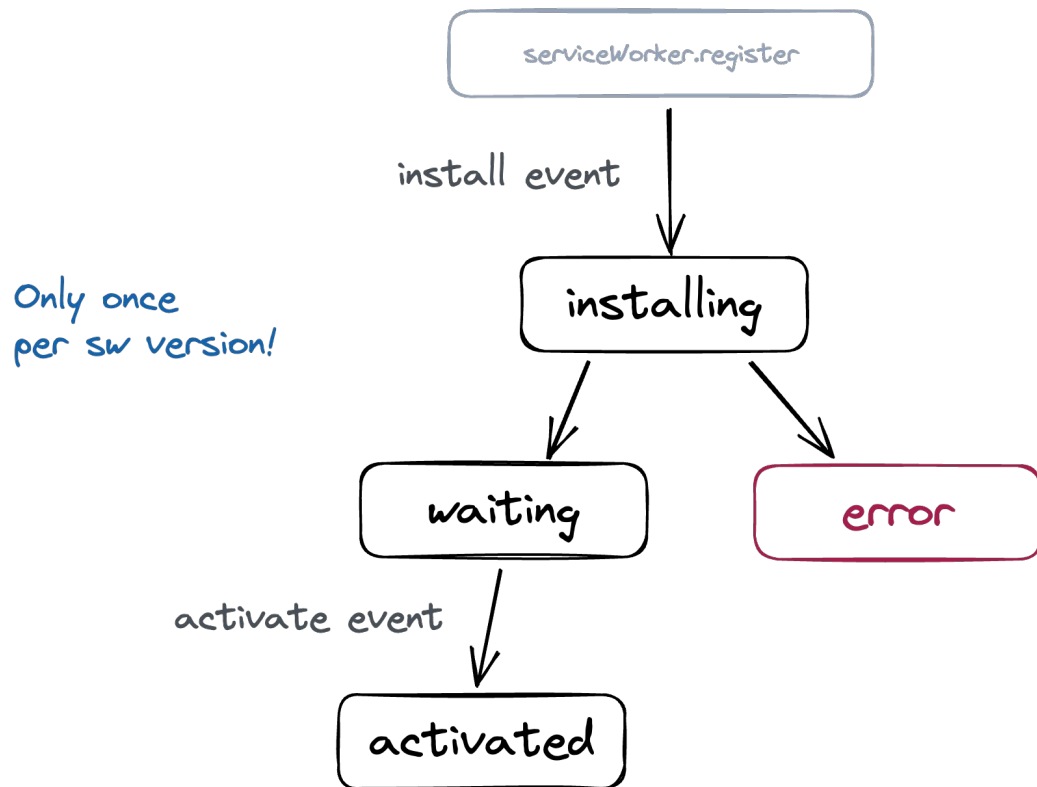
- Installs the service worker and gives it control of the page.



sw.js

```
1 if ("serviceWorker" in navigator) {  
2     navigator.serviceWorker.register("sw.js");  
3 }
```

The service worker lifecycle



Service worker scope determines the pages it can control

- By default, the service worker is allowed to control any paths within the directory it located in.

`www.example.com/scripts/sw.js`

`controls` → `www.example.com/scripts/mypage`

`does not control` → `www.example.com/`

- Only one service worker per scope is allowed to be installed at a time!

`www.example.com/sw.js` ← `conflicts with above`

- In most cases, the service worker file should be in the root directory so it can control the whole site.

Demo 1 — Lifecycle & updating

The fetch event

- The service worker can listen for the fetch event to be notified when a network request happens, and can build a custom response object to respond the request.



sw.js

```
1 self.addEventListener("fetch", (event) => {  
2   console.log(`Handling event for ${event.request.url}`);  
3   if (shouldHandleRequest) {  
4     event.respondWith(responsePromise);  
5   }  
6   // If we do not call `event.respondWith`, the browser  
7   // will continue to handle the request as if no service  
8   // workers were present.  
8 });
```

Ways we can build a response object

- Make a network request using fetch and return the result
- Create a new response object from static data, IndexedDB, or other source
- Using a response stored in a cache using the Cache API



sw.js

```
1 self.addEventListener("fetch", (event) => {  
2   if (handleEvent) {  
3     event.respondWith(fetch(event.request));  
4   }  
5 });
```


The Cache API

- Long-term storage solution for **Request** and **Response** object pairs
- Available from both the Service Worker and browser contexts
- Amount that can be stored depends on the browser
- You are responsible for cleaning up any unneeded cache entries




```
1 const cache = await caches.open("cache_key");  
2 const isPresent = await caches.has("cache_key");  
3 const wasDeleted = await caches.delete("cache_key");  
4 const keys = await caches.keys();
```

Putting items into the cache



```
1 cache.add("/main.js");  
2 cache.add(new URL("/main.js"));  
3 cache.add(new Request("/main.js"));  
4  
5 cache.addAll(["/main.js", "/", "index.html"]);
```

Retrieving items from the cache



```
1 const response = await cache.match("/main.js", {
2   ignoreSearch: false,
3   ignoreMethod: false,
4 });
5
6 if (response) {
7   // Value found!
8 } else {
9   // Value not found in cache
10 }
```

Responding to the fetch event from the cache



```
1 self.addEventListener("fetch", (event) => {  
2   event.respondWith(handleRequest(event.request));  
3 });  
4  
5 async function handleRequest(request) {  
6   const cache = await caches.open("myCache");  
7   const response = await cache.match(request);  
8  
9   // Use the cache match if present  
10  if (response) return response;  
11  
12  // If no match, use the network to make the request  
13  return fetch(request);  
14 }
```


Demo 2 — Precaching files

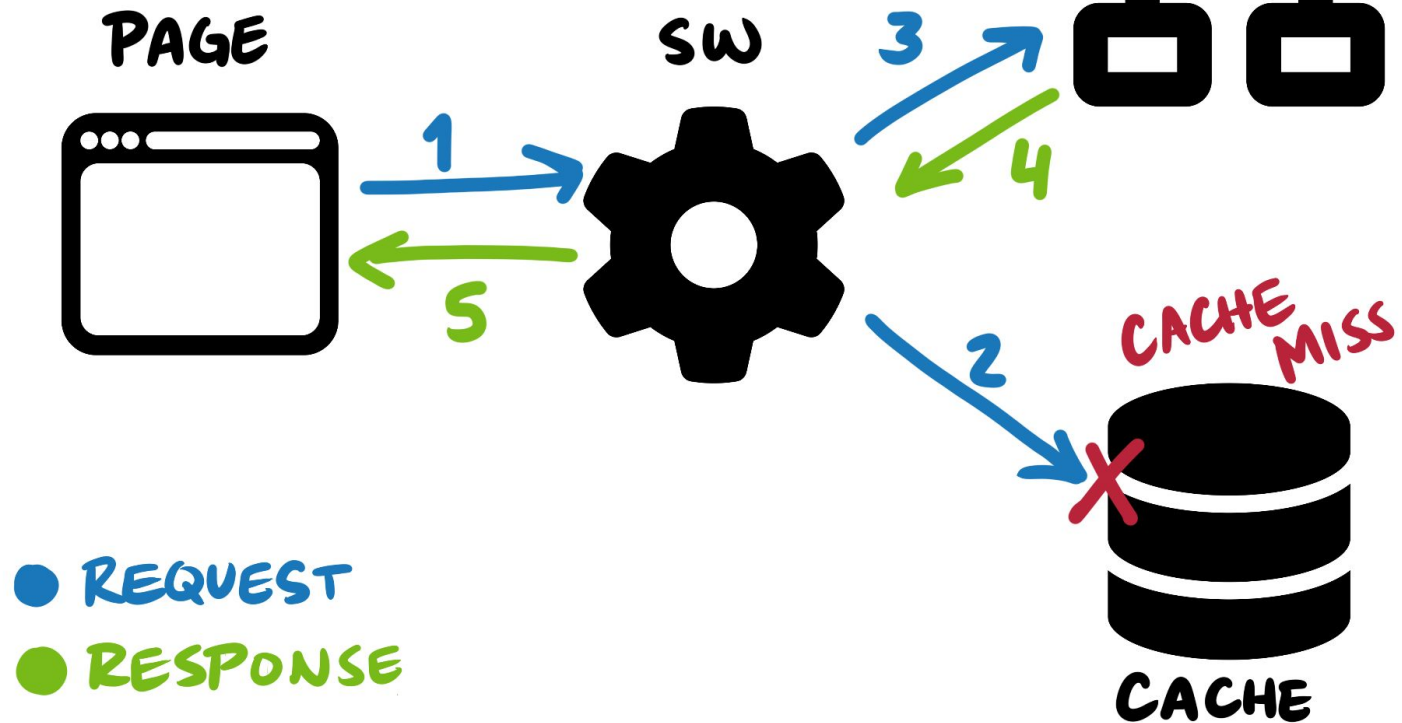
Adding responses to the cache dynamically



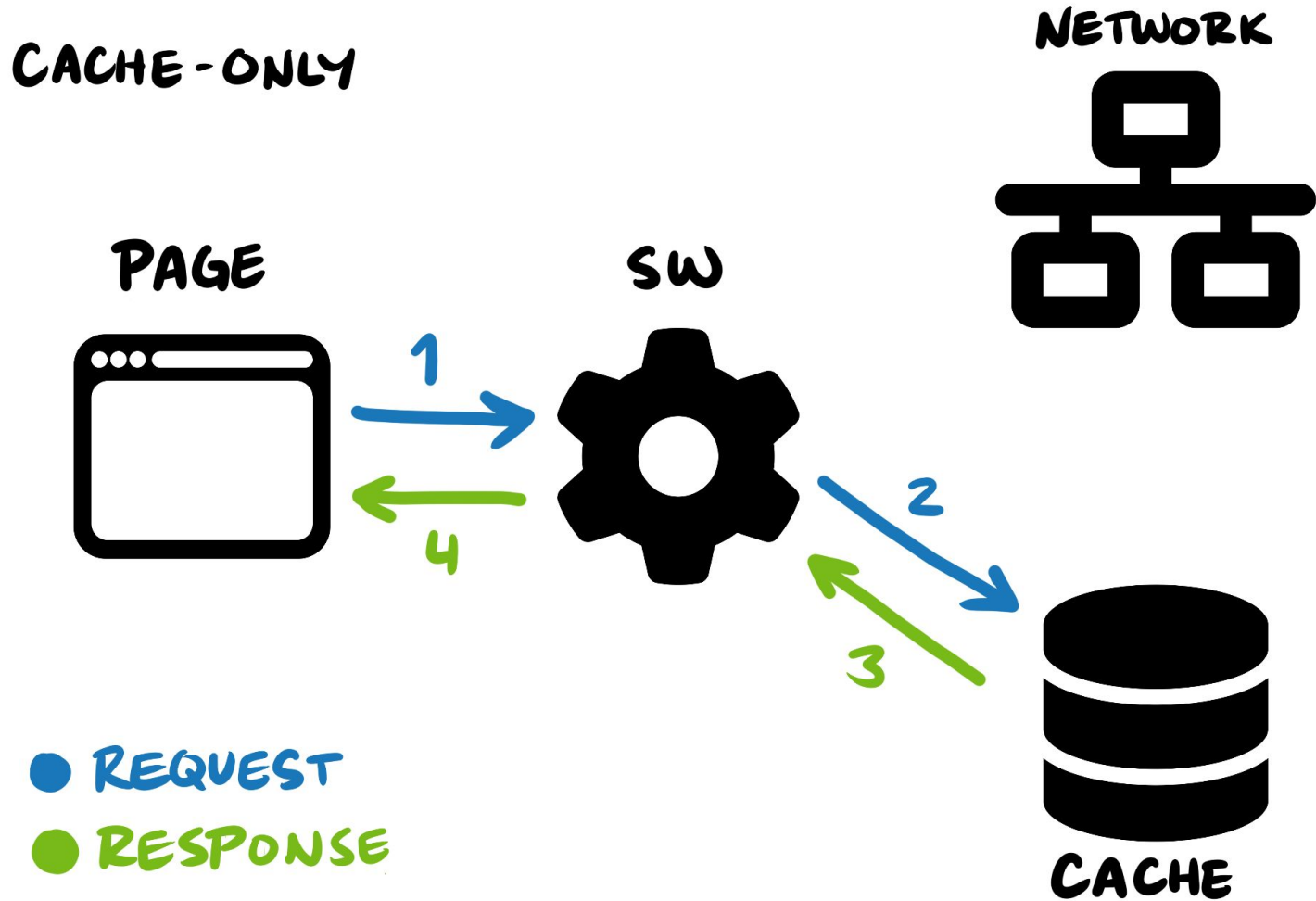
A code editor window with a title bar containing three colored circles (red, yellow, green) and a filename 'sw.js'.

```
1 const handleRequest = async (request) => {  
2   if (/\. (jpg|png|gif)/.test(request.url)) {  
3     const response = await fetch(request);  
4     const cache = await caches.open("myCache");  
5     cache.put(request, response);  
6     return response;  
7   }  
8  
9   return fetch(request);  
10 };
```

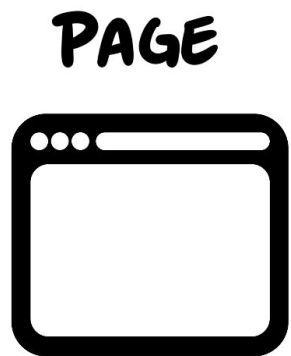
CACHE - FALLING BACK TO NETWORK



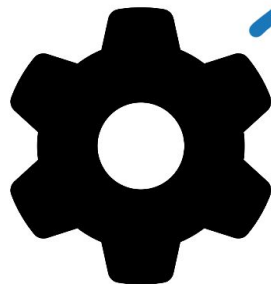
CACHE-ONLY



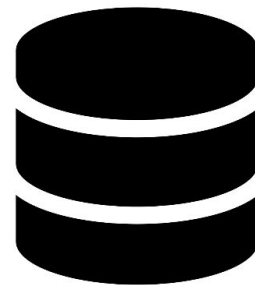
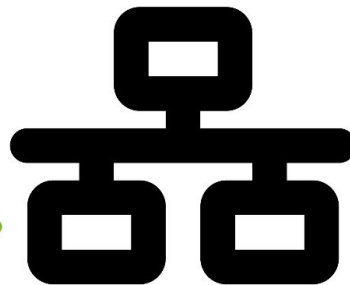
NETWORK ONLY



SW

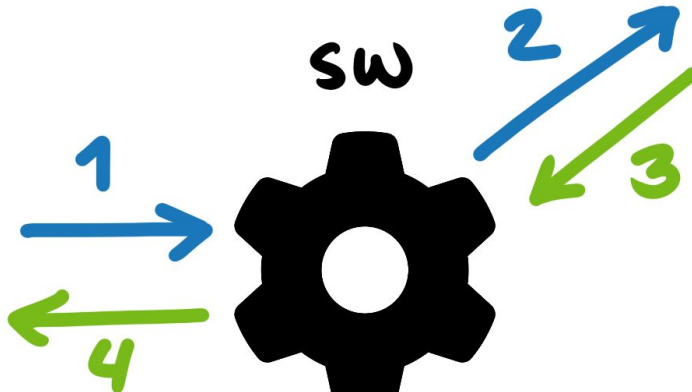


NETWORK



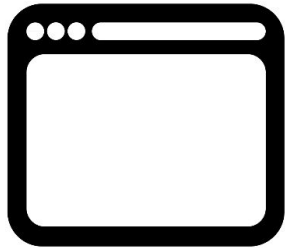
CACHE

- REQUEST
- RESPONSE

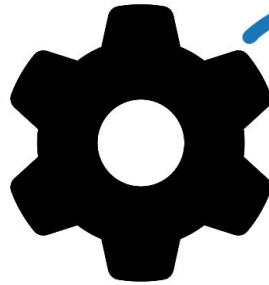


NETWORK - FALLING BACK TO CACHE

PAGE



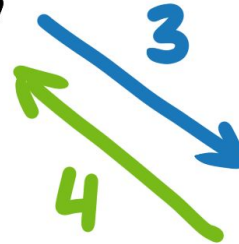
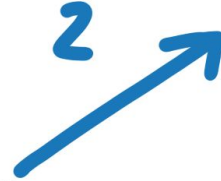
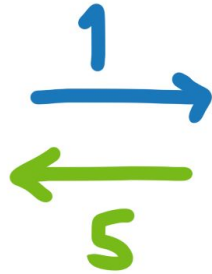
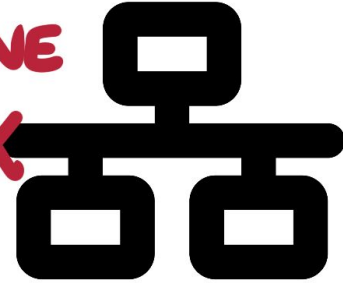
SW



NETWORK

OFFLINE

X



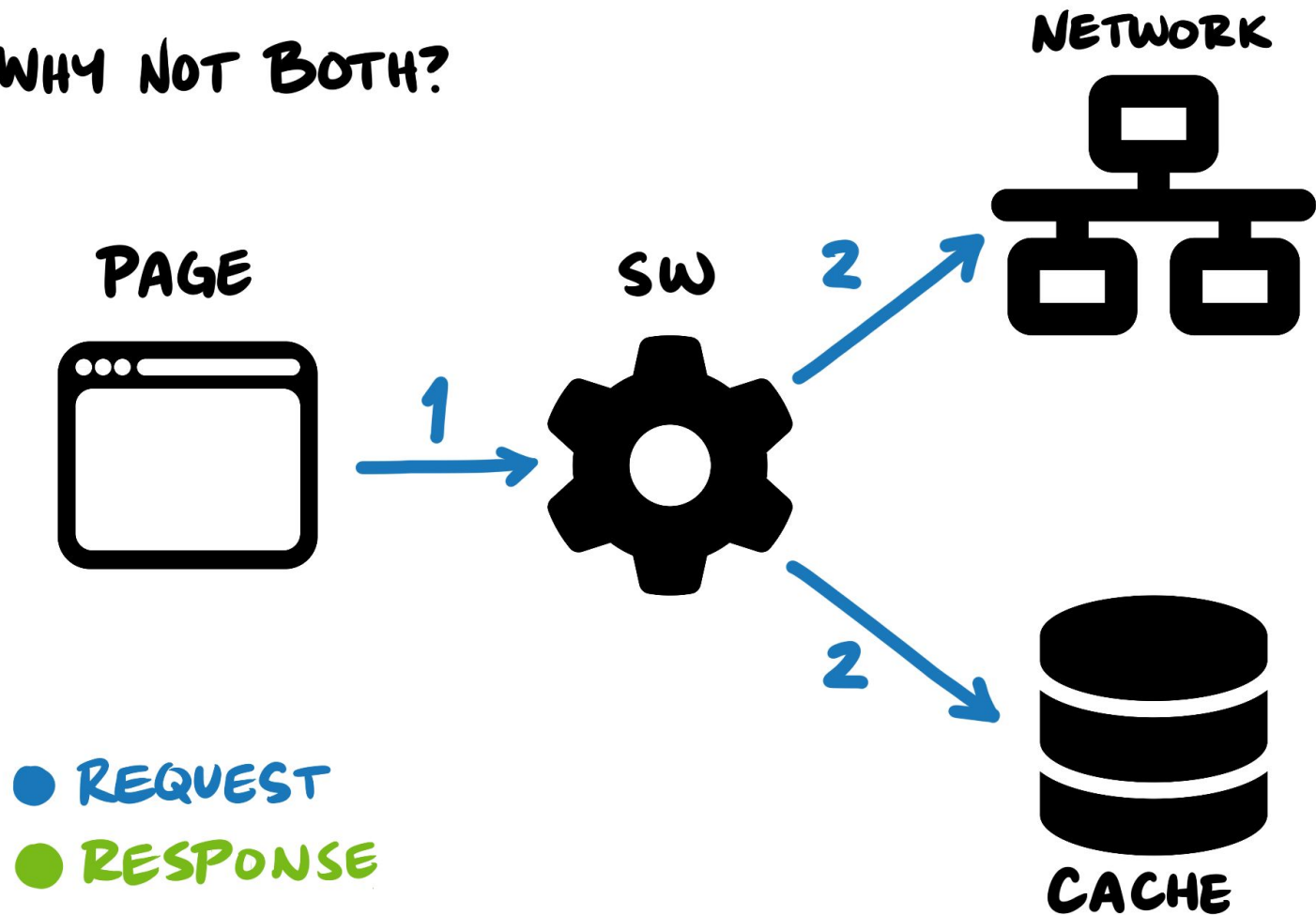
● REQUEST

● RESPONSE

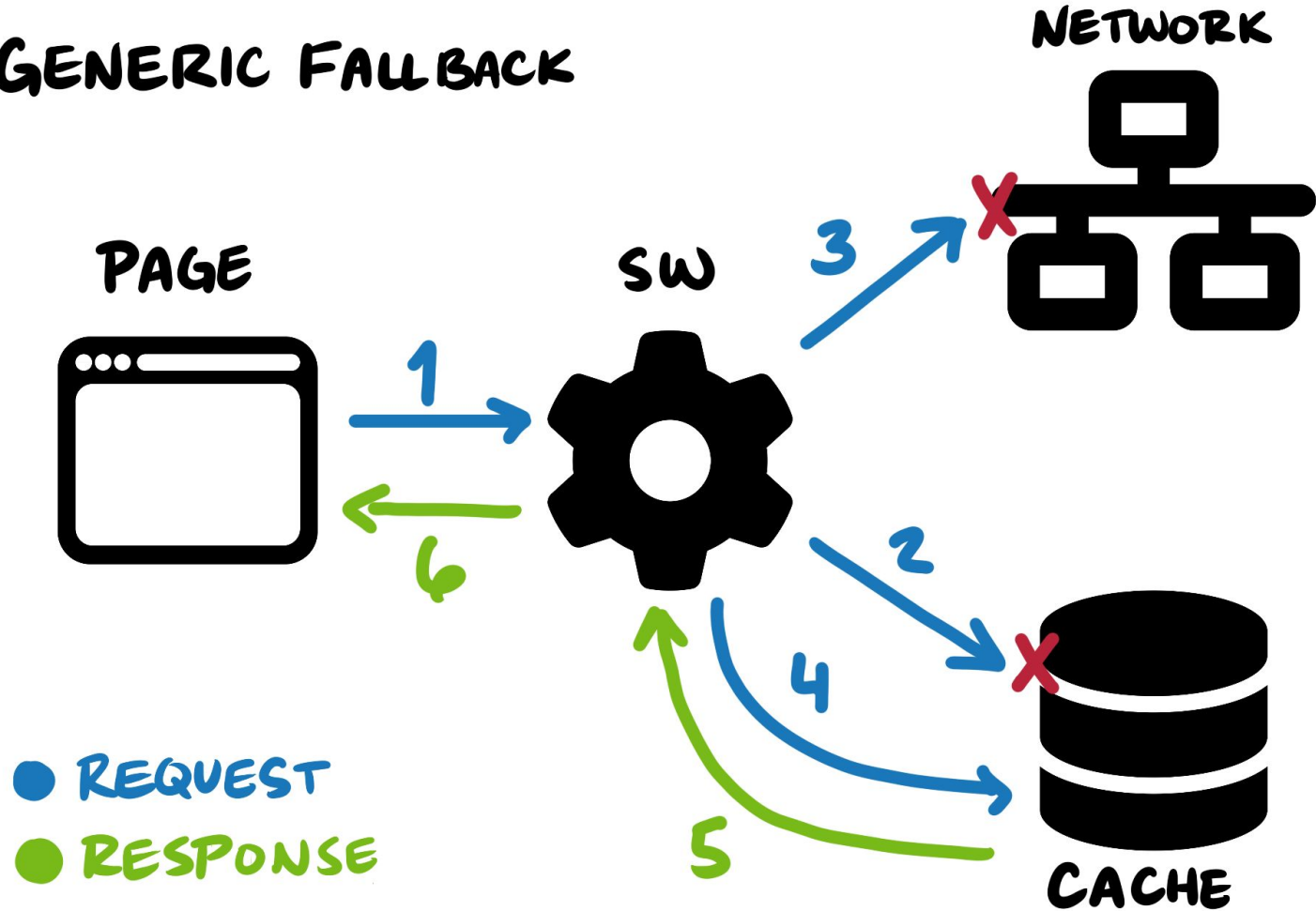


CACHE

WHY NOT BOTH?



GENERIC FALLBACK



Demo 3 — Applying caching strategies

Using service workers in your project

- Google's Workbox toolset is commonly used to make more advanced service worker patterns easy to use.
- Plugins available for many build tools and frameworks to automatically cache built assets with little or no configuration:
 - vite-plugin-pwa
 - workbox-webpack-plugin
 - rollup-plugin-workbox
 - gatsby-plugin-offline
 - next-pwa
 - nuxt-pwa

Things to keep in mind when starting to use caching

- Consider how frequently the items you are caching might change
 - Does the filename change when the asset changes?
 - How can we clean up stale assets?
- How much are we caching?
 - This impacts how much storage is needed, and how long it takes to install the service worker if you are precaching.
 - Rule of thumb — Precache the “app shell”, including any files that are needed to run the site. Cache other things like images at runtime.
- Is it worth the performance impact?
 - Using service workers for network events has a small performance impact and may not be appropriate when comparing it to the standard browser cache

Where things can go wrong

- Unless you are explicitly working on the service worker, do not register it in development
 - You can unregister a service worker from the developer tools to get yourself out of this situation
- Do not store state in variables in the service worker — it is not guaranteed to remain active between events
 - It will always stay active when your developer tools are open, which may be misleading about how it will work in production

Where things can go wrong

- Changing the service worker filename is tricky — can use a self-destroying service worker if absolutely needed
- Caching more complex files like audio and video (which are usually downloaded in chunks with range requests) can be challenging.
 - Google has an demo of how to do this with a service worker + IndexedDB with source code available here: <https://kinoweb.dev/>
- Making cross-origin requests in no-cors mode results in an opaque response, which means we cannot see if the response we are caching was successful or failed
 - By default, workbox and other libraries will ignore these sorts of responses
 - Using CORS allows proper caching of cross-origin requests

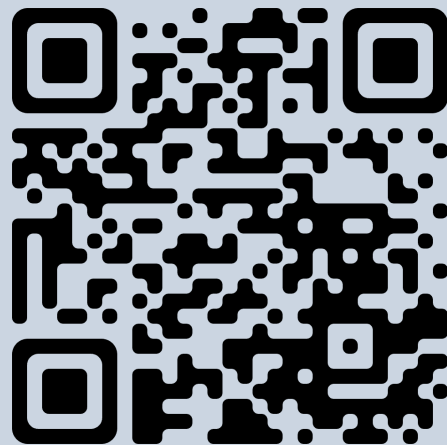
Other tips and tricks

- Developing multiple sites with service workers locally? If you are using Chrome or Firefox, you can use `foo.localhost` to have unlimited domains pointing to `localhost`
- Force reloading the page will cause the browser to bypass the service worker and be uncontrolled. But be careful with this during development!
- Chrome dev tools
 - `chrome://serviceworker-internals/?devtools` – See all of the service workers that are installed on your machine with extra debug info
 - `chrome://inspect/#service-workers` – See all of the currently running service workers, and open their devtools in a new window

Kat Fairbanks

Full stack developer
Cincinnati community organizer

`kat@craftsmanship.dev`
`@derkatzenbar`



Github repo with resources