Google

Progressive Web Apps

Reliable

Going Offline

How could an in-browser proxy help make your PWA work offline?

Going Offline - The Key Players

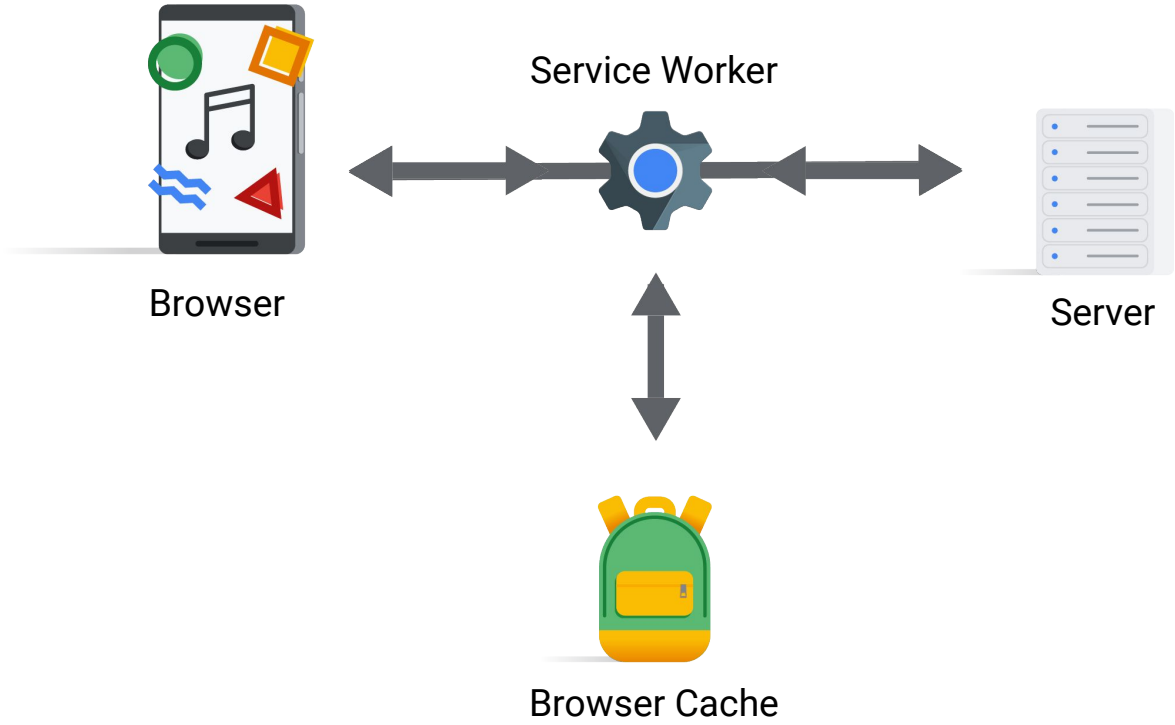
Service Workers

- In-browser proxy
- Works online or offline

Cache Storage API

- Manage browser's site cache
- Add/remove/delete/check for cache items

Life of a Proxied Request



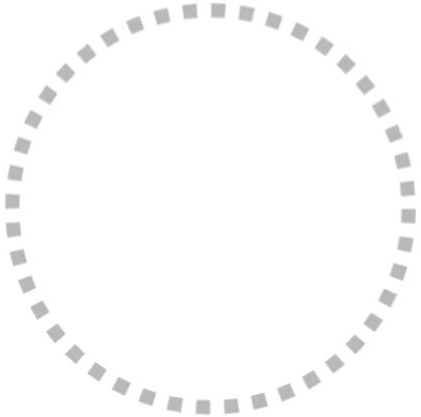
Service Worker Scope - `https://example.com/app/sw.js`

URL	In-Scope	Reason

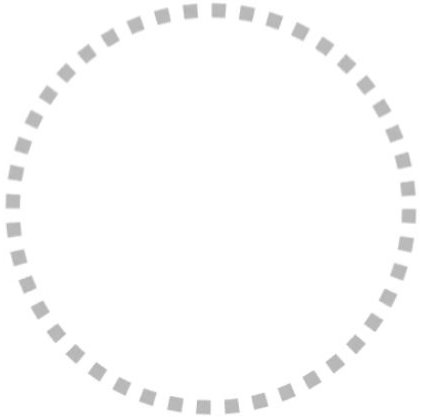
Service Worker Update - `https://example.com/app/sw.js`

URL	Content Changed	Update	Reason

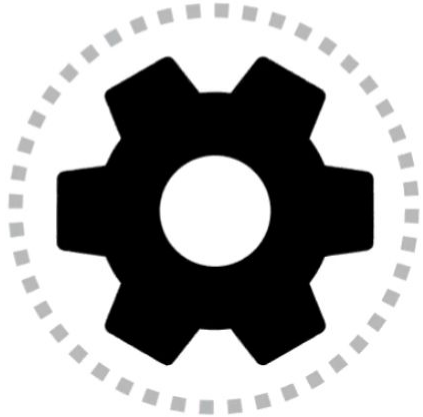
Installing



Waiting



Active

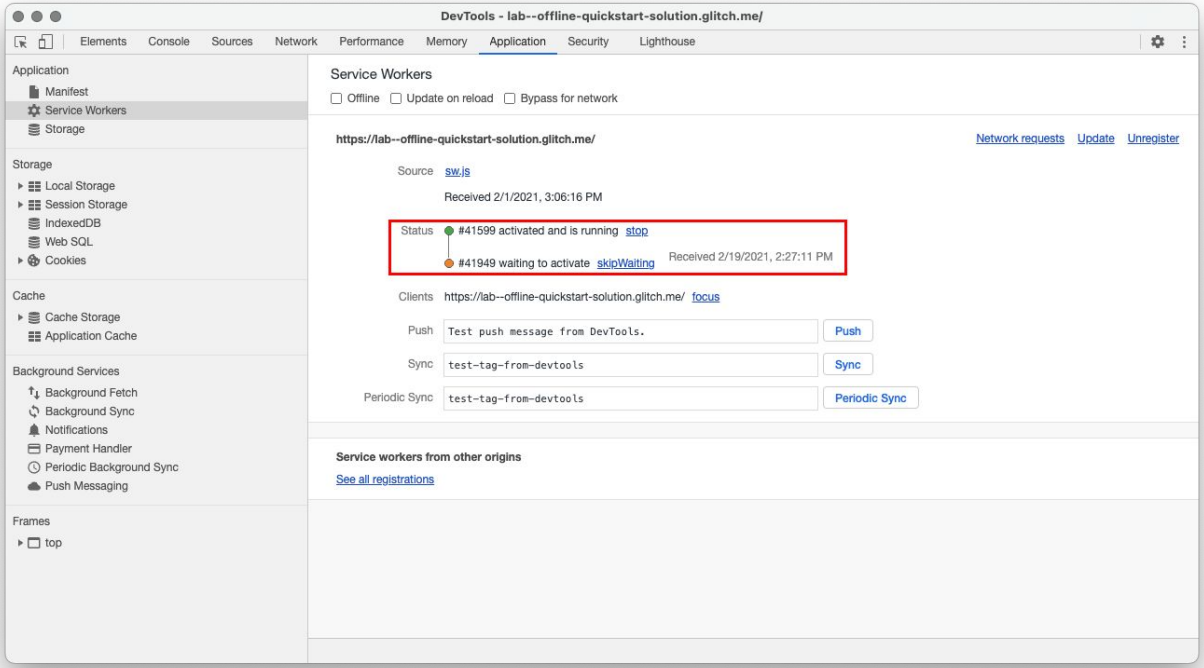


Service Workers & Cache Storage API

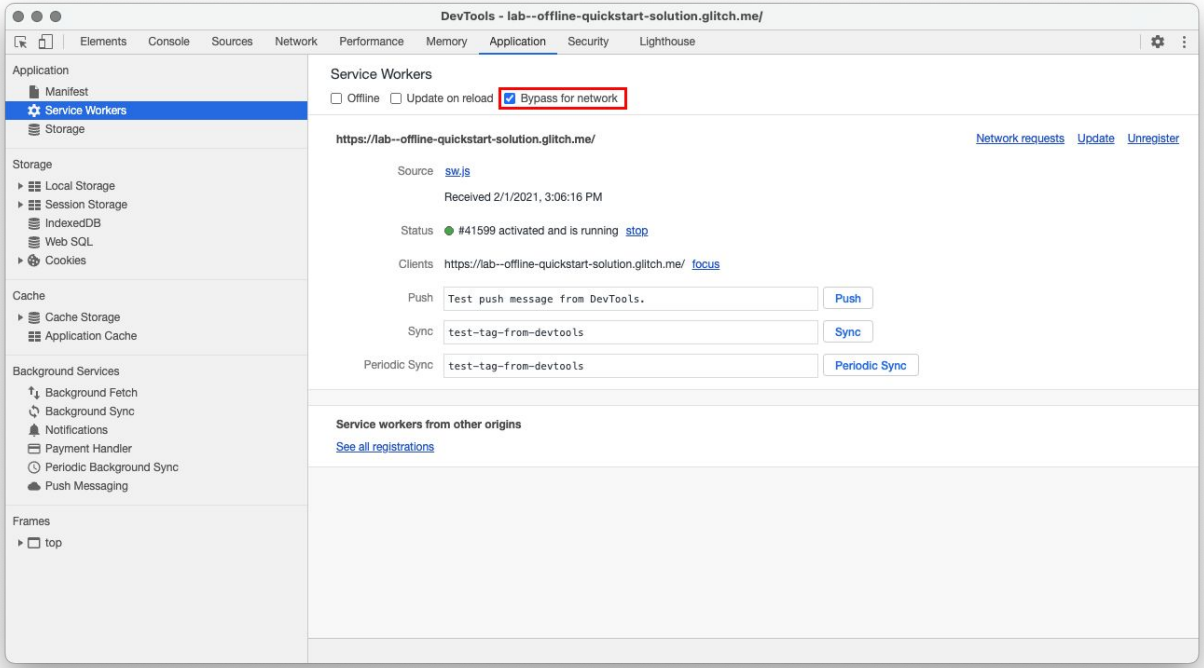
Debugging with DevTools

Google

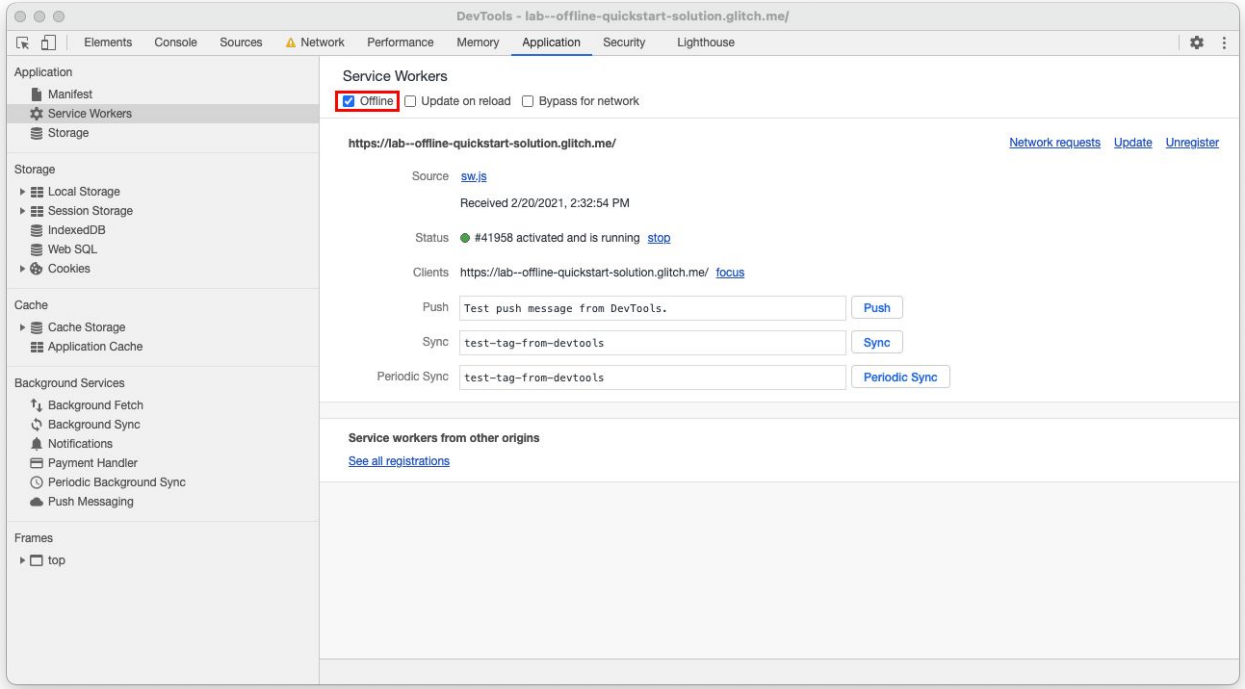
Application Panel - Service Workers



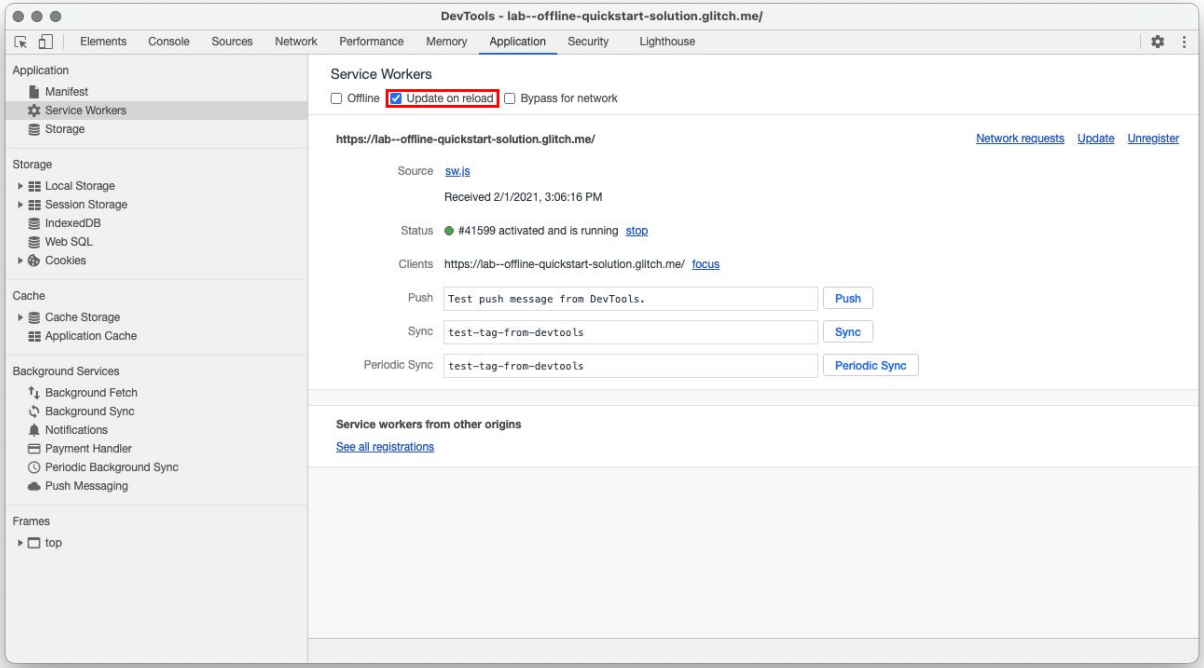
Application Panel - Service Workers



Application Panel - Service Workers



Application Panel - Service Workers



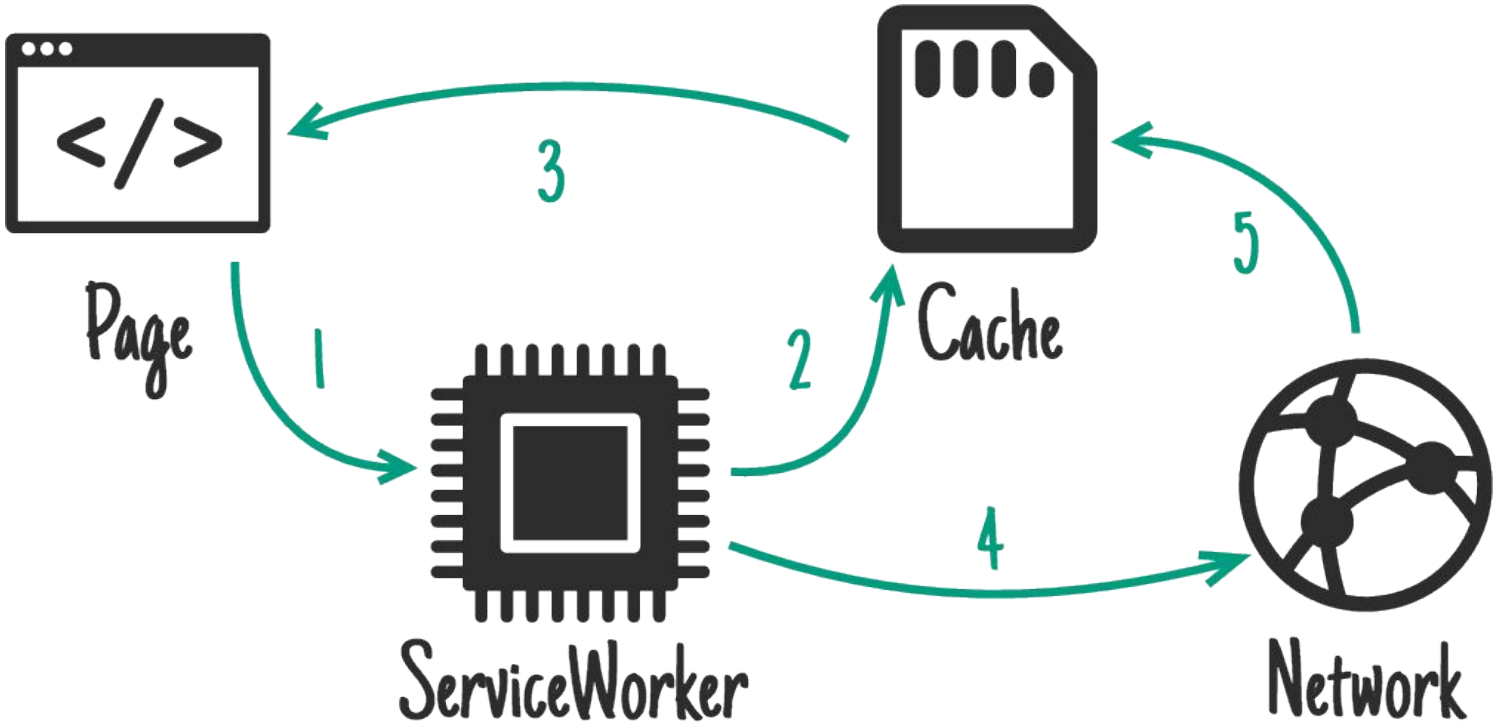
Try It Out - 15 Minutes

https://workshops.page.link/pwa03--going-offline

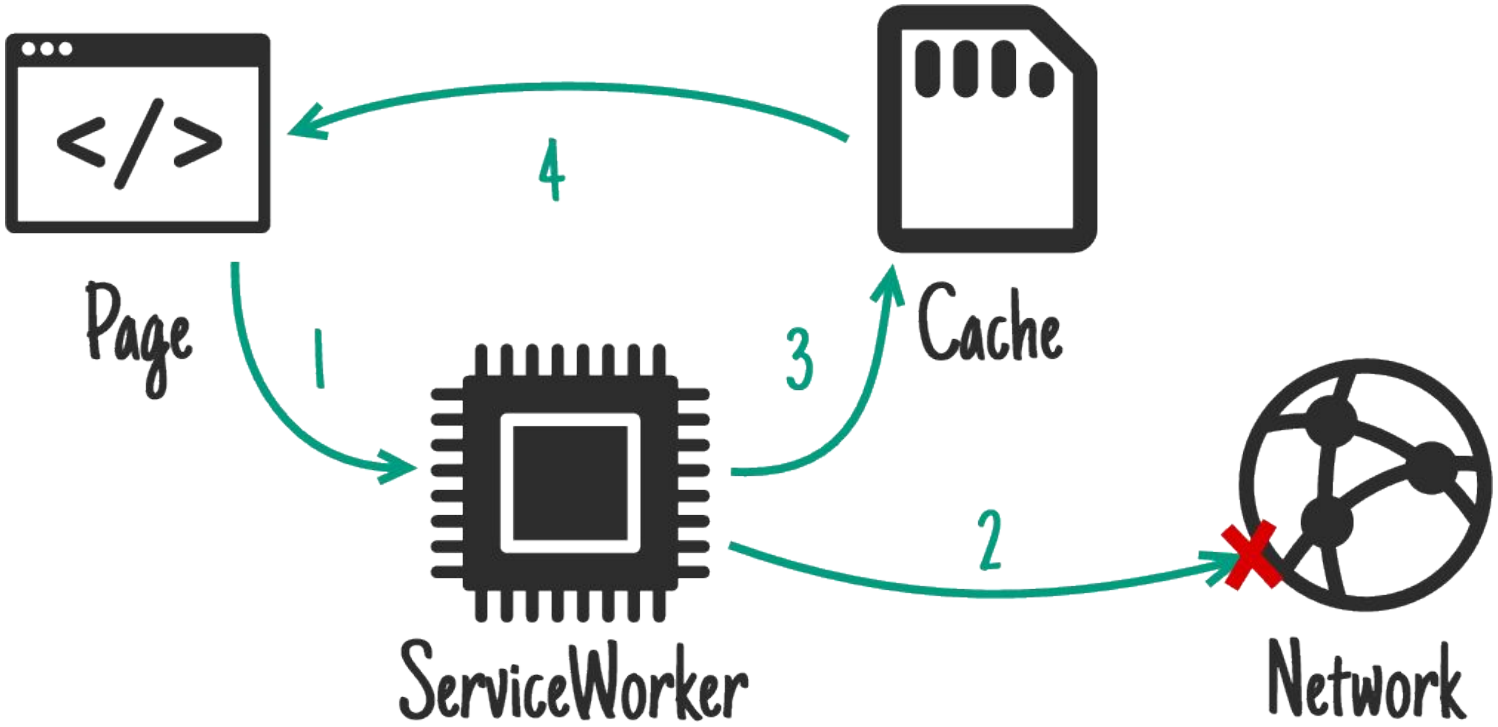
Caching Strategies

How might you cache frequently updated content? Infrequently updated?

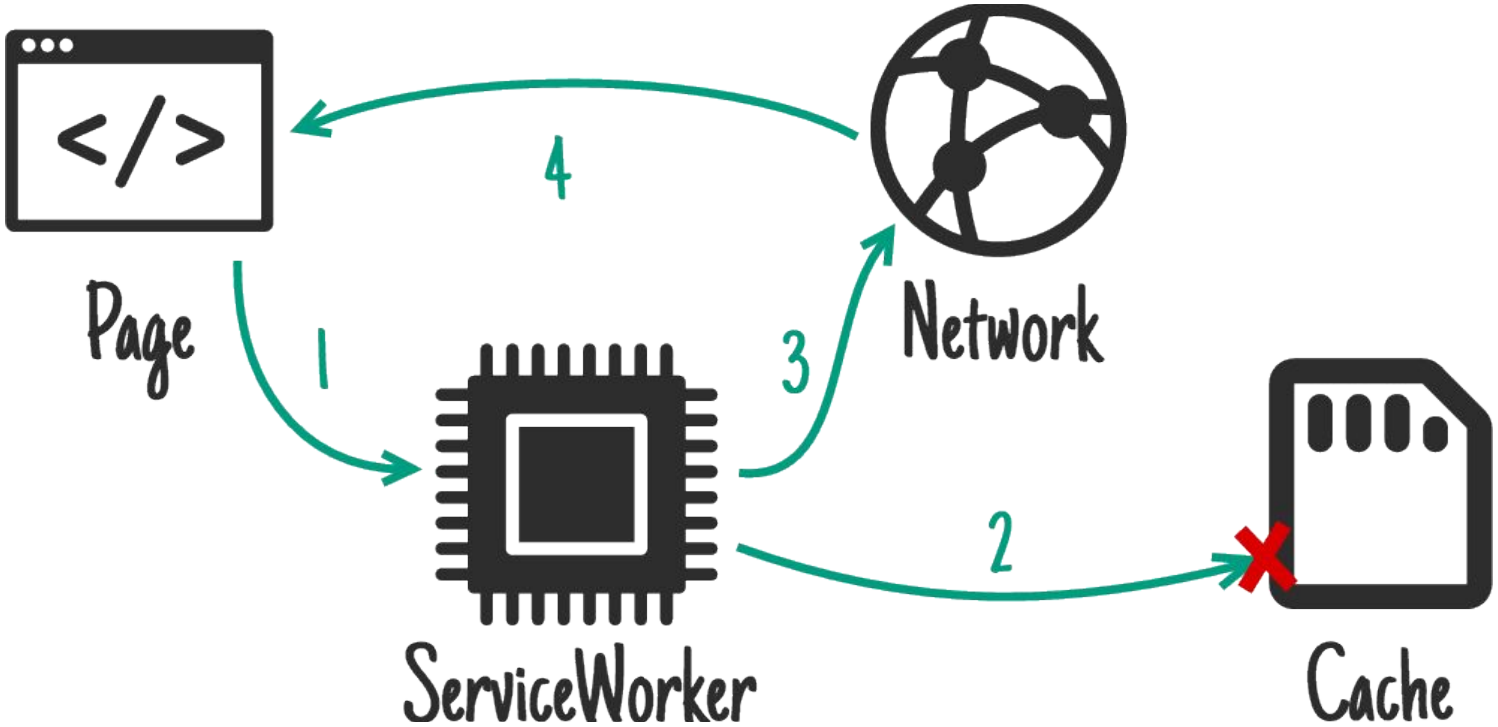
Caching Strategy - Stale-While-Revalidate



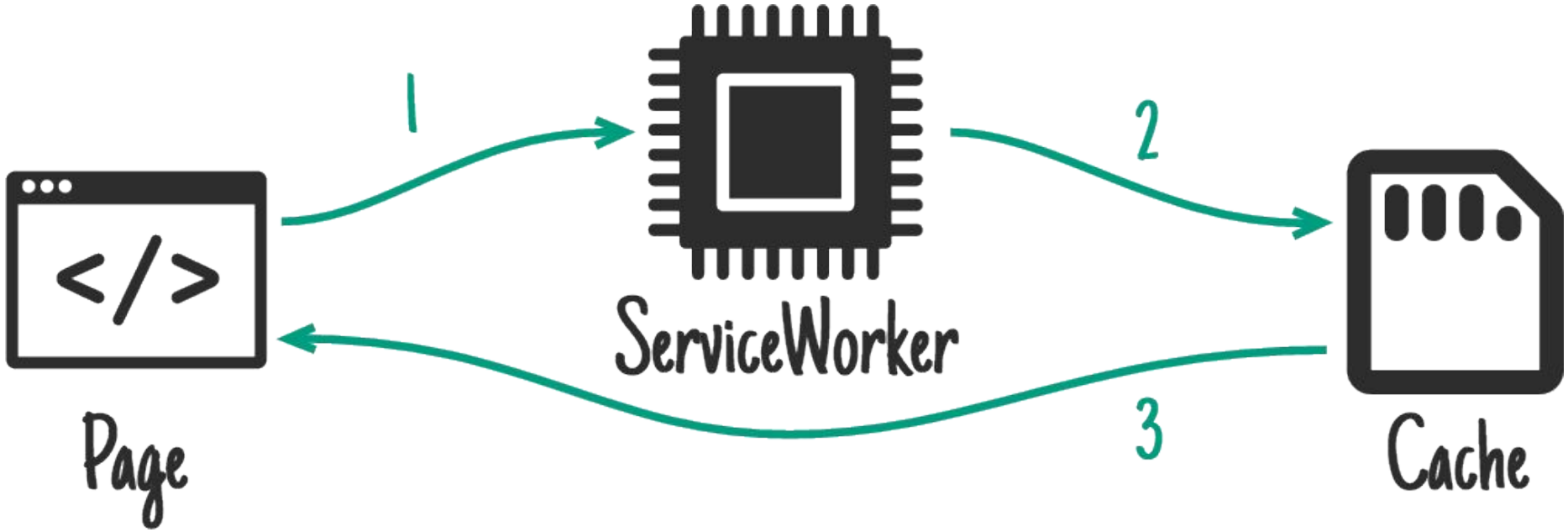
Caching Strategy - Network First



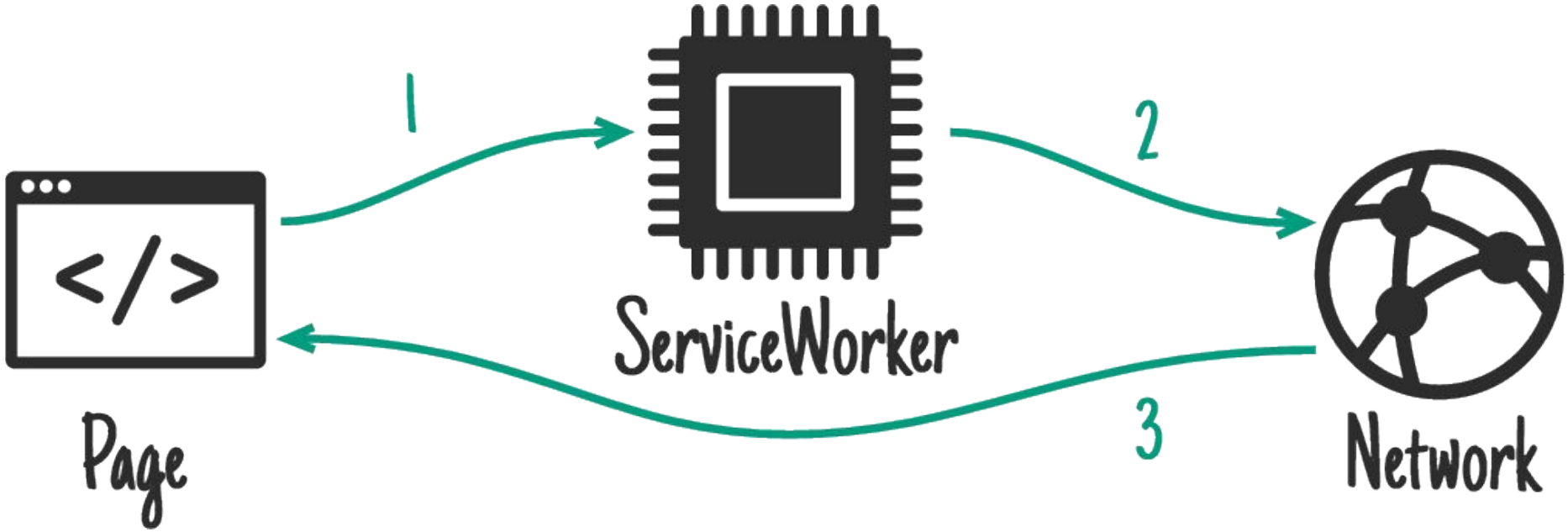
Caching Strategy - Cache First



Caching Strategy - Cache Only



Caching Strategy - Network Only



Choosing a Strategy

- How frequently is this updated?
- Does this depend on something else?

Working with Workbox

Workbox - Setting Up Routing and Caching

```
import { registerRoute } from 'workbox-routing';
import { NetworkFirst } from 'workbox-strategies';
import { CacheableResponsePlugin } from 'workbox-cacheable-response';
```

Workbox - Setting Up Routing and Caching

```
const pageStrategy = new NetworkFirst({
  // Put all cached files in a cache named 'pages'
  cacheName: 'pages',
  plugins: [
    // Ensure that only requests that result in a 200 status are cached
    new CacheableResponsePlugin({
      statuses: [200],
    }),
  ],
});
```

Workbox - Setting Up Routing and Caching

```
// Cache page navigations (html) with a Network First strategy
registerRoute(
  // Check to see if the request is a navigation to a new page
  ({ request }) => request.mode === 'navigate',
  // Use the strategy
  pageStrategy
);
```

Workbox - Offline Fallback

```
import { setCatchHandler } from 'workbox-routing';

// Warm the cache when the Service Worker installs
self.addEventListener('install', event => {
  const files = ['/offline.html'];

  event.waitUntil(
    self.caches.open('offline-fallbacks')
    .then(cache => cache.addAll(files))
  );
});
```

Workbox - Offline Fallback

```
// Respond with the fallback if a route throws an error
setCatchHandler(async (options) => {
  const dest = options.request.destination;
  const cache = await self.caches.open('offline-fallbacks');

  if (dest === 'document') {
    return (await cache.match('/offline.html')) || Response.error();
  }

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

Workbox Recipes

```
import {
  pageCache,
  googleFontsCache,
  offlineFallback,
} from 'workbox-recipes';

pageCache();
googleFontsCache();

offlineFallback({
 pageFallback: '/offline.html'
});
```

Try It Out - 15 Minutes

https://workshops.page.link/pwa03--workbox

IndexedDB

How do you think IndexedDB differs from Cache Storage?

Cache Storage API

- ✓ Network resources
- ✓ File-based content

IndexedDB

- ✓ Structured data
- ✓ Searchable data

Opening a Database

```
// Using https://github.com/jakearchibald/idb
import { openDB } from 'idb';

const db = openDB('bands', 1, {
  upgrade(db, oldVersion, newVersion, transaction) {
    // Switch over the oldVersion to allow the database to be incrementally upgraded. No
`break` so all of the updates get run!
```

Opening a Database

```
  switch(oldVersion) {
    case: 0
      // Placeholder to execute when database is first created (oldVersion is 0)
    case 1:
      // Create a store of objects
      const store = db.createObjectStore('beatles', {
        // The `name` property of the object will be the key.
        keyPath: 'name'
      });
      // Create an store index called `age` based on the `age` key of objects in the store
      store.createIndex('age', 'age');
  }
 }
});
```

The “Beatles” Object Store

#	Key (Key path: “name”)	Value
0	John Lennon	`{name: 'John Lennon', nickname: 'The smart one', age: 40, living: false}`
1	Paul McCartney	`{name: 'Paul McCartney', nickname: 'The cute one', age: 73, living: true}`
2	Ringo Starr	`{name: 'Ringo Starr', nickname: 'The funny one', age: 74, living: true}`

Getting an Item

```
// Using https://github.com/jakearchibald/idb
const tx = await db.transaction('beatles', 'read')
const store = tx.objectStore('beatles');
const value = await store.get('John Lennon');
```

Getting an Item

```
// Using https://github.com/jakearchibald/idb
const george = {
  name: 'George Harrison',
  nickname: 'The shy one',
  age: 58,
  living: false
};
const tx = await db.transaction('beatles', 'readwrite');
const store = tx.objectStore('beatles');
store.add(george);
await tx.done
```

The “Beatles” Object Store

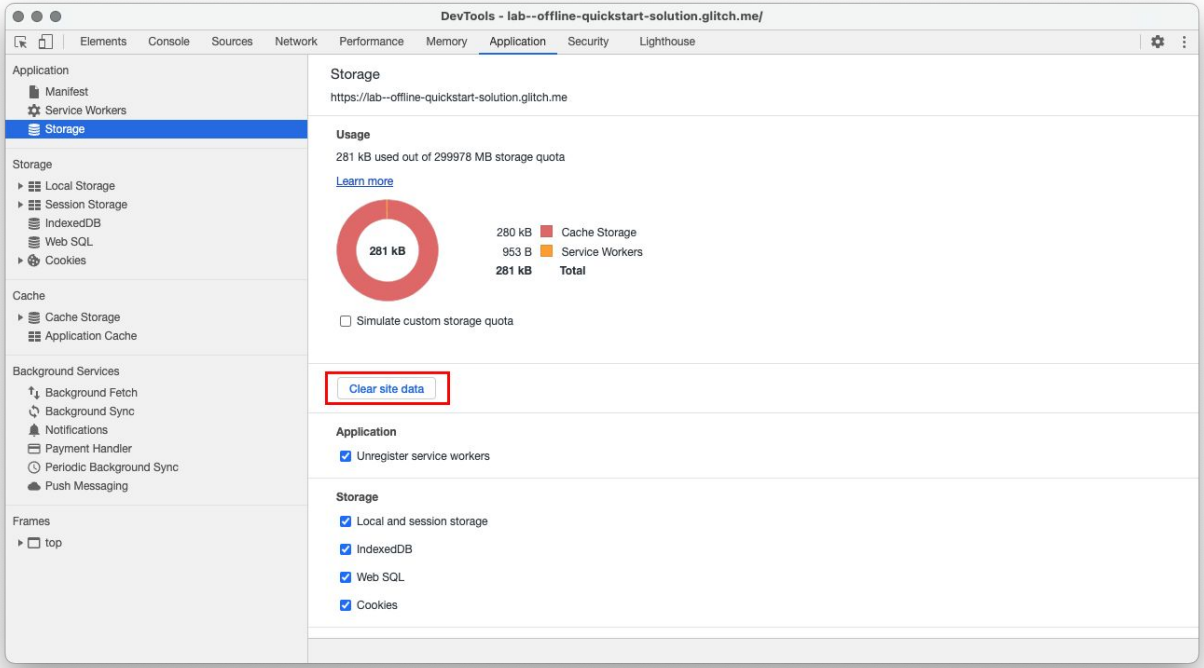
#	Key (Key path: “name”)	Value
0	John Lennon	`{name: 'John Lennon', nickname: 'The smart one', age: 40, living: false}`
1	Paul McCartney	`{name: 'Paul McCartney', nickname: 'The cute one', age: 73, living: true}`
2	Ringo Starr	`{name: 'Ringo Starr', nickname: 'The funny one', age: 74, living: true}`
3	George Harrison	`{name: 'George Harrison', nickname: 'The shy one', age: 58, living: false}`

Managing Storage

Available Storage

```
if (navigator.storage && navigator.storage.estimate) {
  const quota = await navigator.storage.estimate();
  // quota.usage -> Number of bytes used.
  // quota.quota -> Maximum number of bytes available.
  const percentageUsed = (quota.usage / quota.quota) * 100;
  console.log(`You've used ${percentageUsed}% of the available storage.`);
  const remaining = quota.quota - quota.usage;
  console.log(`You can write up to ${remaining} more bytes.`);
}
```

Storage in the Applications Panel



Try It Out - 15 Minutes

https://workshops.page.link/pwa03--indexeddb

```
await tx.done;
```