

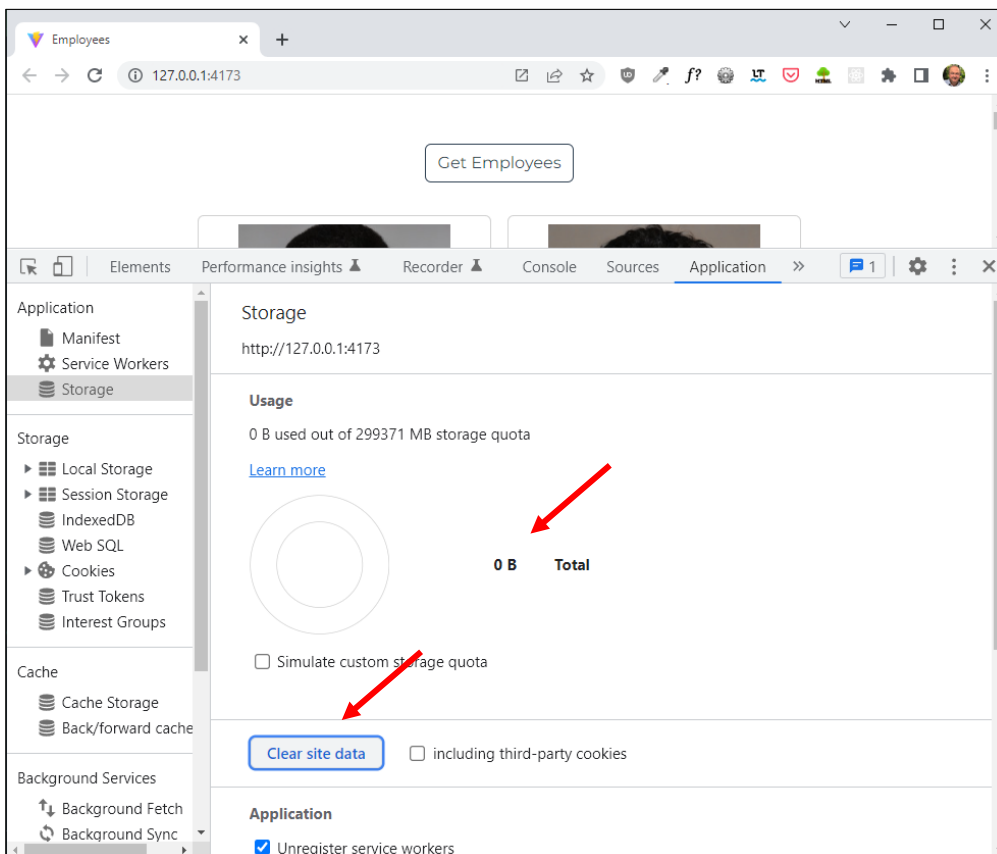
Robert Baumgartner

Wir setzen mit dem letzten Stand der App aus dem vorigen Arbeitsblatt fort. Dort haben wir schon einiges über Service Worker erfahren. Mit diesem Wissen sehen wir uns nun den Service Worker genauer an.

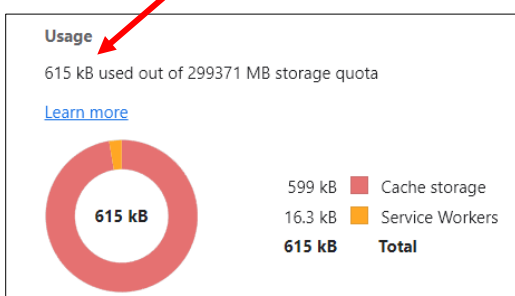
Aufgabe 1: Stelle das Port für den Preview Server von 5555 auf das Defaultport 4173 um. Entferne aus **vite.config.js** von den Optionen für VitePWA das Property **registerType**.

`registerType: 'autoUpdate'` bewirkt, dass – auch wenn ein Service Worker bereits aktiv ist – der neue Service Worker ohne Rücksicht auf Verluste den alten Service Worker ersetzt. Für das Development mag das praktisch sein, in Production kann jedoch der Benutzer Daten verlieren. Daher ist es besser eine Meldung einzublenden, dass eine neue Version der App vorliegt und den Benutzer entscheiden zu lassen (siehe weiter unten).

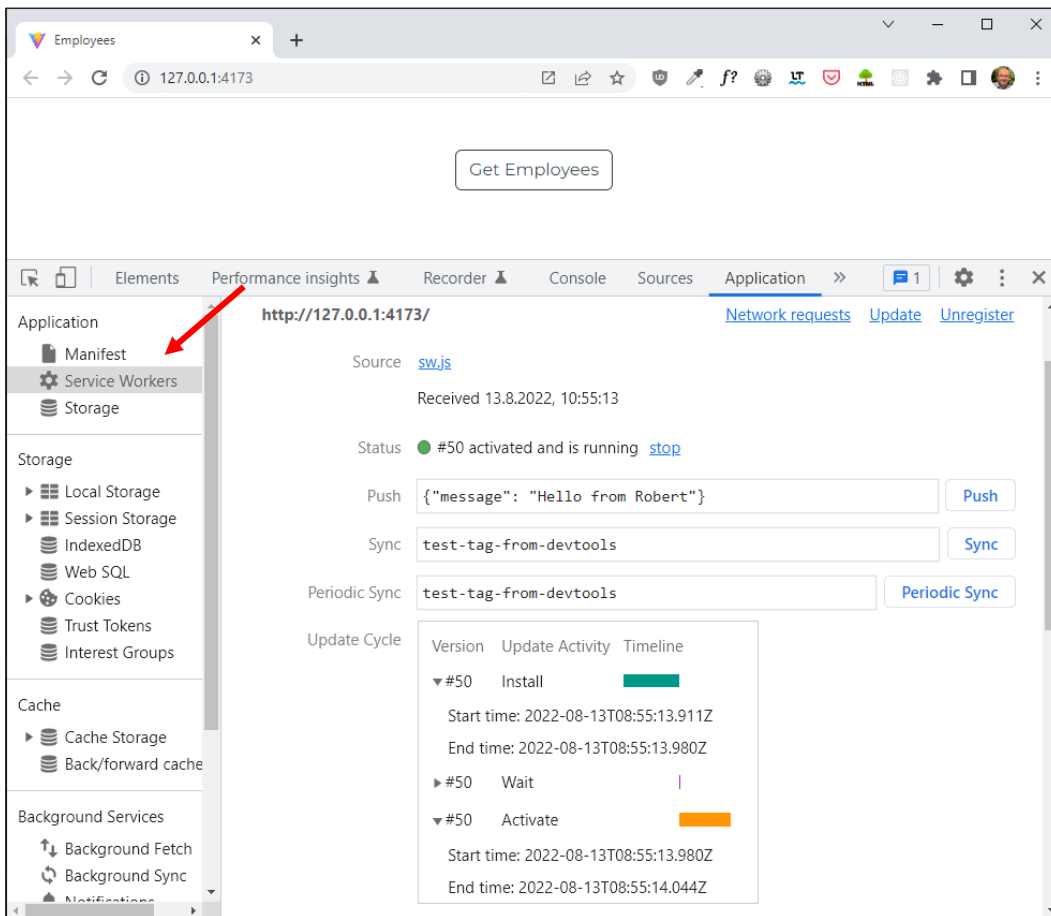
Aufgabe 2: Starte deine App (Build und Preview) und lösche in den Dev Tools unter **Application/Storage** den lokalen Speicher für die Origin **http://127.0.0.1:4173**. Eine Origin ist definiert als eine Kombination aus Protokoll, Hostname und Portnummer. Alles, was durch die Ausführung eines JavaScript Programms lokal gespeichert wird (Cookies, Service Worker, Cache, localStorage, etc.) ist der jeweiligen Origin von der das Script geladen wurde, zugeordnet. Daher wird durch das Löschen der Storage einer Origin auch ein eventuell gespeicherter (und aktiver) Service Worker entfernt.



Lade die Seite neu und beobachte, wie der Speicherverbrauch gestiegen ist.



Wähle in der linken Leiste **Service Workers** aus.

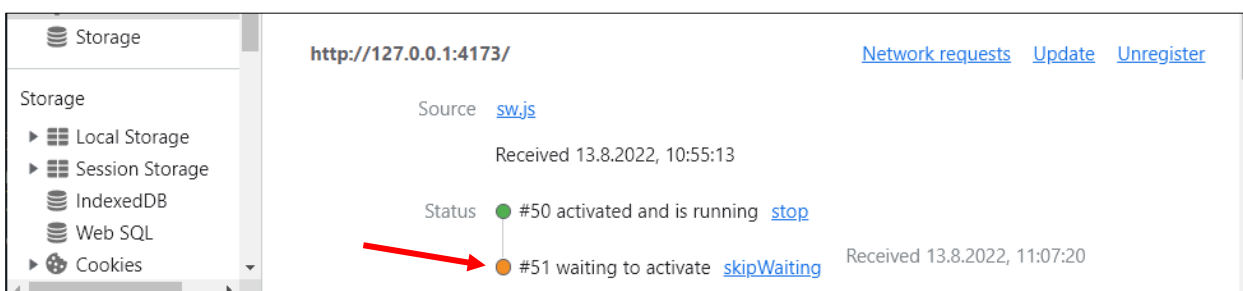


Du siehst: Der Service Worker ist aktiv und läuft. Des Weiteren siehst du den Update Zyklus: **install**, **wait** und **activate**. Bevor der Service Worker jedoch installiert werden kann, muss er zuerst registriert werden. Das erledigt das Script **registerSW.js** für uns, das in **index.html** eingebunden ist.

Aufgabe 3: Füge nun einen h3 Tag zu **App.vue** hinzu!

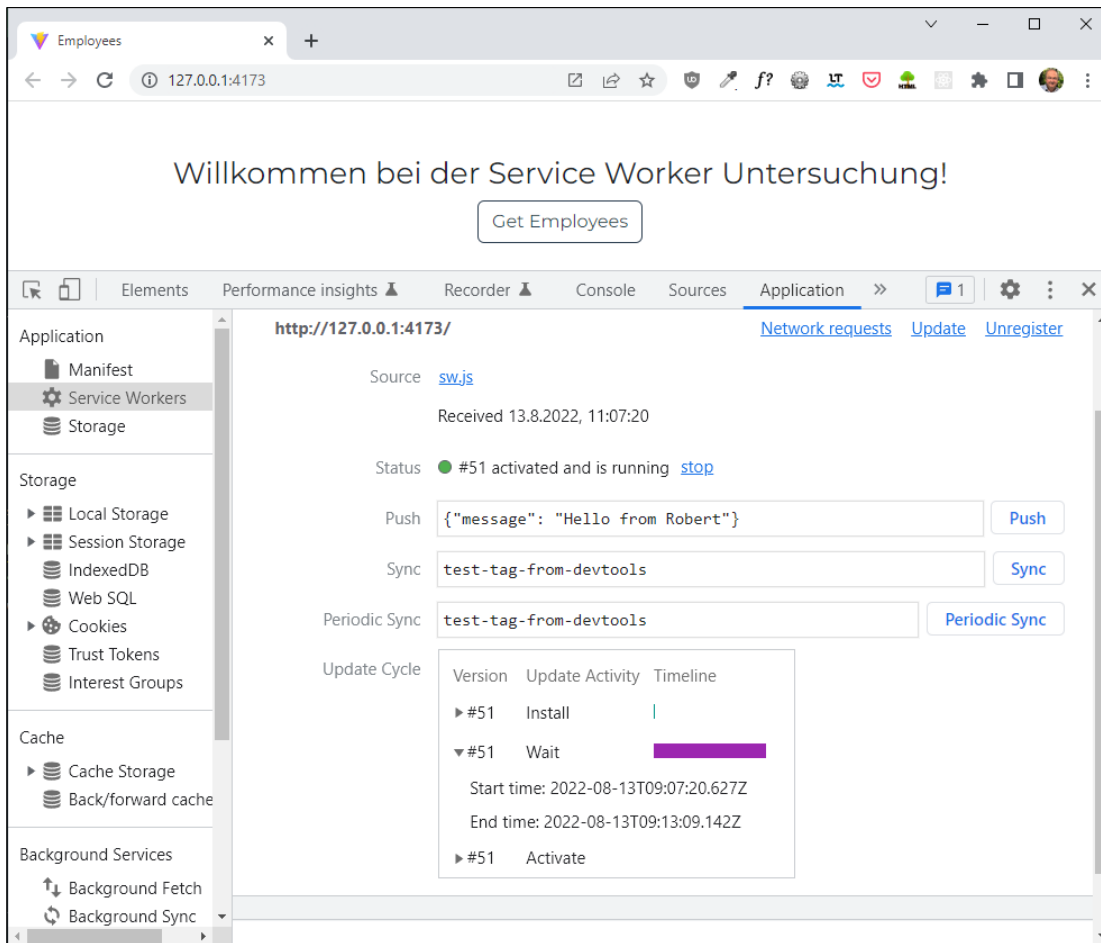
```
<h3>Willkommen bei der Service Worker Untersuchung!</h3>
<ButtonGet @get="fetchData"></ButtonGet>
<CardView :employees="employees" @del="delEmployee"></CardView>
```

Schließe den Browser nicht! Baue erneut einen Build und lade die Seite neu. Nichts ändert sich für den Benutzer. Was ist der Grund dafür? Die App wird aus dem Cache geladen. Der neue Service Worker ist jedoch schon registriert, installiert und wartet darauf, aktiv zu werden. Da wir jedoch **autoUpdate** entfernt haben, muss das jetzt manuell passieren. Ein Refresh nützt nichts.

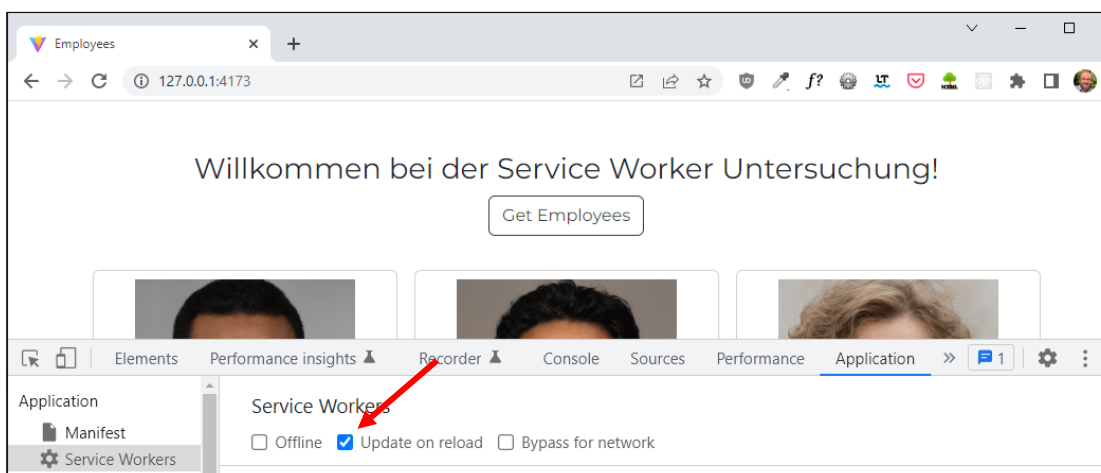


Robert Baumgartner

Klicke beim Service Worker, der wartet, auf den Link **skipWaiting**. Der alte Service Worker wird verworfen. Lade die Seite neu. Der neue Service Worker wird jetzt aktiv. Die neue Version erscheint!



Der Service Worker wird erst ersetzt, wenn der Browser (alle Tabs!) geschlossen ist und neu gestartet wird. In den Dev Tools gibt es mehrere Möglichkeiten, den gleichen Effektas manuell herbeizuführen. **skipWaiting** ist eine Möglichkeit. **Update on reload** eine andere!



OK. Es wird Zeit, sich den Lebenszyklus eines Service Workers genauer anzusehen. Doch zunächst ein paar wichtige Konzepte. Behalte dazu im Hinterkopf, dass der Service Worker ein Netzwerk-Proxy ist.

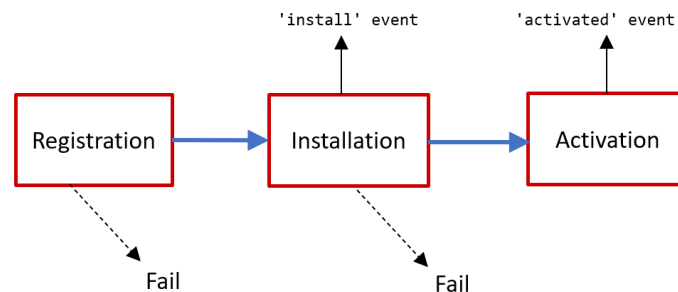
Kontrolle und Scope

Wenn wir sagen, ein Service Worker kontrolliert eine Seite, dann handelt es sich um eine Seite, bei der ein Service Worker Netzwerkanfragen für diese Seite abfängt. Der Service Worker ist aktiv **und** in der Lage, innerhalb eines bestimmten Geltungsbereiches (Scopes) für die Seite zu agieren, i.e. Anfragen aus dem Cache zu beantworten, oder Push Nachrichten zu empfangen.

Der Geltungsbereich eines Service Workers wird dadurch definiert, wo sich sein Code auf einer Website befindet (bei uns ist das üblicherweise das **/public** Verzeichnis, was also Root (**/**) entspricht. Wenn ein Service Worker sich jedoch im Unterverzeichnis **/example** befindet, ist der Scope des Service Workers nur **/example** (inkl. der Unterverzeichnisse).

Lebenszyklus eines Service Workers bei Erstinstallation

Wenn eine Website einen neuen Service Worker registriert, durchläuft der Service Worker eine Reihe von Zuständen.



Registration

Beim ersten Besuch einer Website (ohne einen registrierten Service Worker), wird gewartet, bis die aktuelle Seite vollständig geladen ist, bevor der Service Worker registriert wird. Das vermeidet Bandbreitenkonflikte, wenn der Service Worker Teile der Applikation **precached**, was praktisch oft der Fall ist.


Precaching = das Laden von Inhalten einer Seite in einen Cache bevor sie benötigt werden, damit, wenn die Seite offline ist, diese aus dem Cache geholt und angezeigt werden können.

Wenn der Inhalt eines Service Workers Syntaxfehler enthält, schlägt die Registrierung fehl und der Service Worker wird nicht registriert. Er wird verworfen.

Wie bereits oben erwähnt: Ein Service Worker wird immer in Bezug auf seine Origin registriert, also in Bezug auf Protokoll, Hostname (Domain) und Port.

Beispiel: Hier wird der Service Worker **sw.js** registriert. Da er sich im Unterverzeichnis **/example** befindet, ist sein Scope auf **/example** beschränkt. Dateien, die im Verzeichnis oberhalb liegen, kann er nicht cachen.

```
window.addEventListener('load', async () => {
  if ('serviceWorker' in navigator) {
    try {
      await navigator.serviceWorker.register('/example/sw.js');
      //...
    } catch (error) {
      console.log(error);
    }
  }
});
```



Robert Baumgartner

Die Registrierung erledigt das Vite PWA Plugin mittels **registerSW.js** für uns. Wir können aber natürlich diesen File überschreiben, wenn wir das möchten.

Wenn die Registration beginnt, geht der Zustand in **'installing'** über.

Installation

Nach erfolgreicher Registrierung wird ein **install** Event ausgelöst. Dieser Event wird nur einmal pro Service Worker aufgerufen und wird erst wieder ausgelöst, wenn der Service Worker aktualisiert wird. Wir können uns für diesen Event registrieren, wenn wir einen eigenen Service Worker erstellen (siehe unten).

Ein neuer Cache wird angelegt und alle Elemente einer Webseite, die preached werden sollen, werden geladen.

Das Vite PWA Plugin verwendet für jegliche Art von Caching **WorkBox** von Google (siehe nächstes Arbeitsblatt). Daher finden sich unsere precached Assets in dem Array, das der WorkBox Funktion **precacheAndRoute** übergeben wird. Die meisten unserer Files sind durch den Build bereits in einem Bundle und alle Files sind mit einem Hashcodes identifiziert. Das passiert durch den Bundler **Rollup**, der von Vite verwendet wird.¹

```
precacheAndRoute(  
  [  
    { url: 'assets/index.008d6c1f.js', revision: null },  
    { url: 'assets/index.ba865d62.css', revision: null },  
    { url: 'index.html', revision: 'fe99327e000fc04786334a33c1653e61' },  
    { url: 'registerSW.js', revision: '1872c500de691dce40960bb85481de07' },  
    { url: 'manifest.webmanifest', revision: 'a682f65f0792ba3d4d2be38f6fde036e' },  
  ],  
  {},  
)
```

WorkBox verwendet den asynchronen Call **waitUntil** zum Anlegen des Caches und dem Herunterladen der Daten. Wenn **waitUntil** erfolgreich beendet ist, geht der Zustand in **'installed'** über. Allerdings kann **waitUntil** auch schiefgehen (Promise rejected). In dem Fall wird der Service Worker verworfen.

Activation

Wenn die Registrierung und die Installation erfolgreich waren, wird der Service Worker aktiviert. Zu dem Zeitpunkt befinden sich bereits alle Files, die precached werden sollen, in ihrem Cache. Der Status ist nun **"activating"**. Es wird ein **activate** Event ausgelöst. Wir können uns für diesen Event registrieren, wenn wir einen eigenen Service Worker erstellen.

Bei neuen Service Workern wird die Aktivierung sofort nach erfolgreicher Installation ausgelöst. Sobald die Aktivierung abgeschlossen ist, wird der Status des Service Workers **'activated'**.

Beachte, dass der neue Service Worker die Seite erst bei der nächsten Navigation oder dem nächsten Reload der Seite kontrolliert.

Lebenszyklus eines Service Workers bei Update

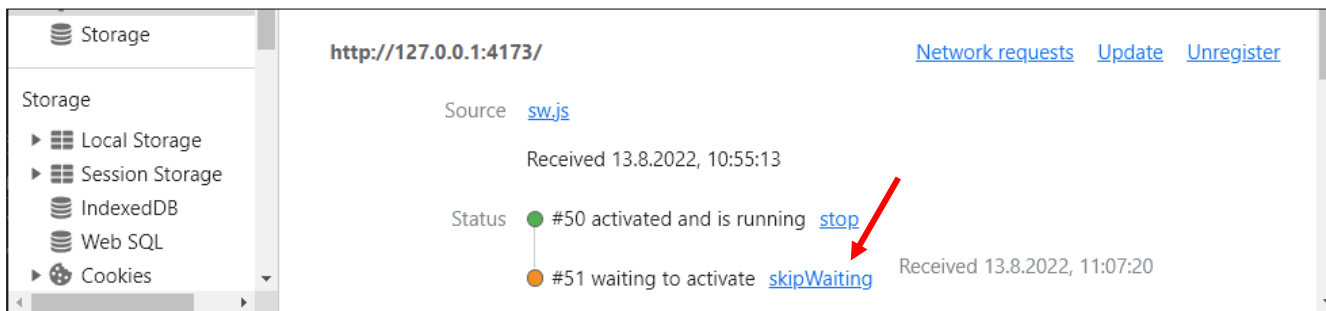
Einmal muss auch der beste Service Worker upgedatet werden. Doch wie bekommt der Browser mit, dass ein neuer Service Worker vorliegt? Zum Beispiel, wenn der Benutzer eine Seite neu lädt (oder Browser neu startet). Oder zu einer Seite navigiert, die im Scope des Service Workers liegt.

¹ Siehe: <https://rollupjs.org/guide/en/>

Installation

Jedes Mal, wenn sich die durch Precache geladenen Daten ändern, ändert sich auch der Service Worker. Das ist etwa bei Änderungen in **/assets** oder **/src** der Fall. Wie wir oben gesehen haben, sind die Inhalte von **/public** per Default nicht im Precache definiert. Daher ändert sich im Fall einer Änderung in **/public** auch nicht der Service Worker.

Doch auch wenn der neue Service Worker installiert ist, ist er noch nicht aktiv. Der alte Service Worker kontrolliert noch die Webseite. Der neue Service Worker kommt in den Zustand **'waiting to activate'**. Standardmäßig wird ein neuer Service Worker aktiviert, wenn keine Clients mehr von dem alten Service Worker kontrolliert werden. Das ist der Fall, wenn alle offenen Tabs für die betreffende Website geschlossen sind. Oder wenn du manuell den alten Service Worker killst.



Aufgabe 4: Nenne alle Methoden, die du kennst, mit denen du manuell das Aktivieren des neuen Service Workers forcieren kannst.

Activation

Wenn ein neuer Service Worker installiert ist und der alte Service Worker keine Seite mehr kontrolliert, wird der neue Service Worker aktiviert. Der Zustand ist **'activated'**.

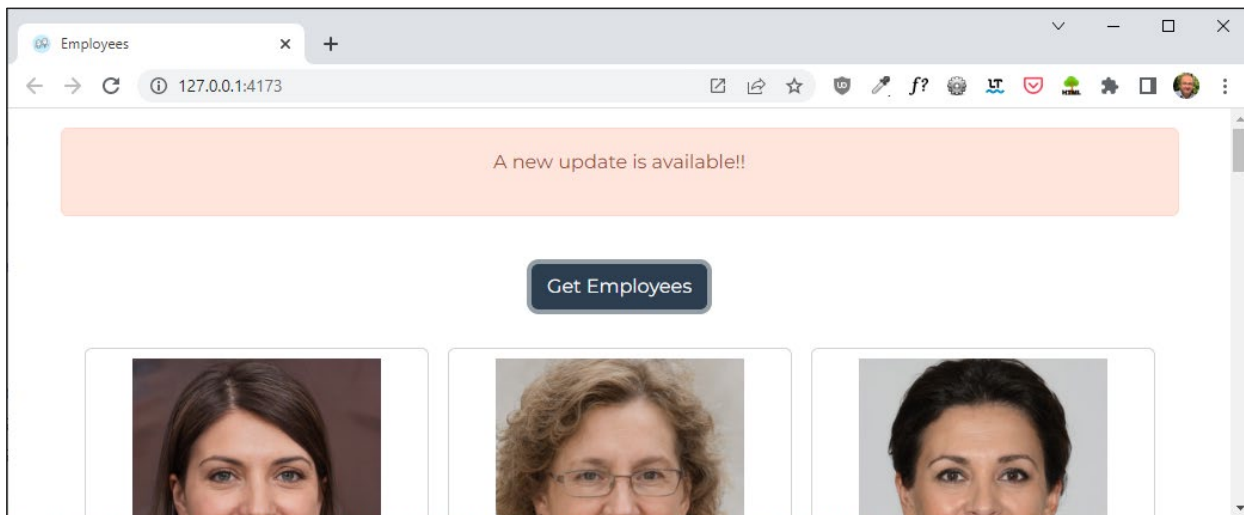
Information bei Update des Service Workers

Genug Theorie. Wir wollen nun eine Update Meldung einbauen, sodass der Benutzer informiert wird, wenn ein neuer Service Worker vorliegt.

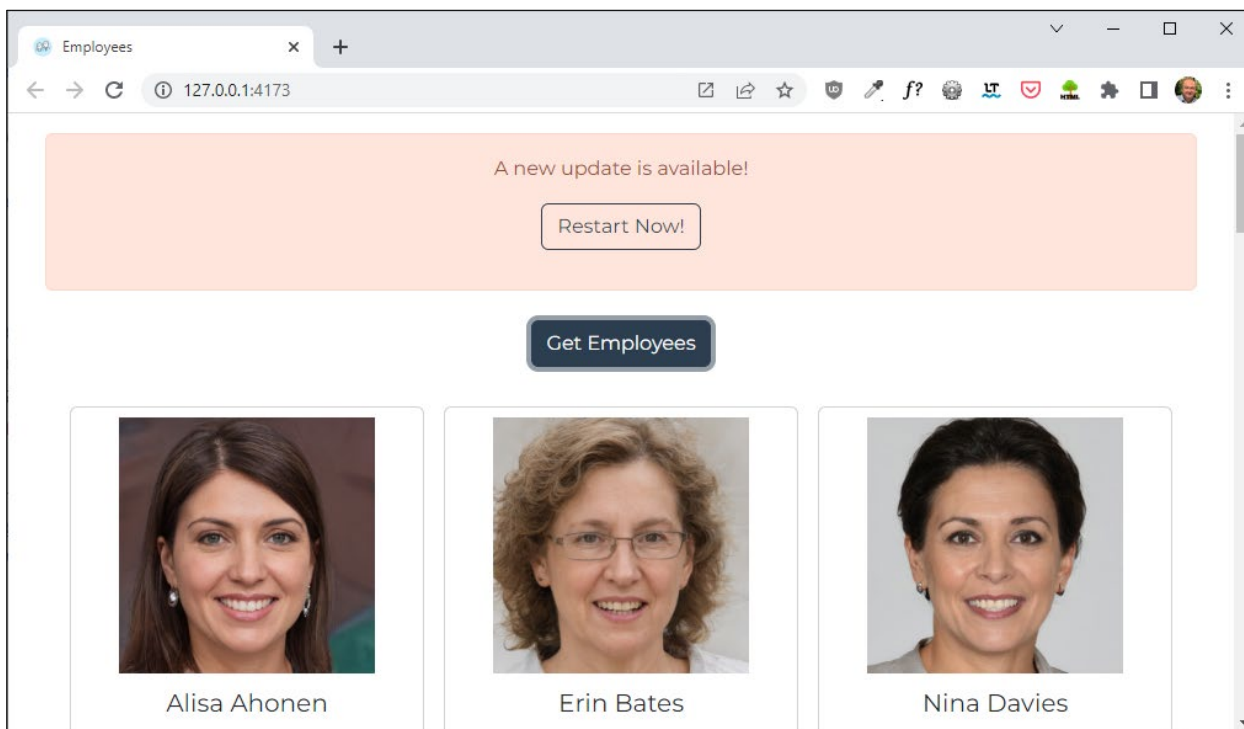
Aufgabe 5: Entferne die Überschrift und verwende/ ergänze dazu folgenden Code in **App.vue** (**update** ist eine Variable, die verwendet werden kann, um das Alert anzuzeigen). Der Code darf das Laden der App nicht blockieren!

```
const registration = await navigator.serviceWorker.getRegistration();
if (!registration) {
  console.log('registration failed!');
  return;
}
registration.addEventListener('updatefound', () => (update.value = true));
if (registration.waiting) update.value = true;
```

Die letzte Zeile ist nötig, da der **updatefound** Event nur einmal gesendet wird. Manche Benutzer refreshen dann einfach die Seite, die Meldung ist weg, aber es wurde nichts geändert! Die Variable **update** wird aber dadurch wieder mit **false** initialisiert und daher ist die Meldung weg. Daher: Falls ein Service Worker **'waiting'** ist, dann bleibt die Anzeige!



Praktisch wäre es wenn der Benutzer die Möglichkeit hätte mittels eines Buttons, den neuen Service Worker zu aktivieren und die Seite neu zu laden



Wir können via Messages seitens der App mit dem Service Worker kommunizieren (und umgekehrt). Der vom Vite PWA Plugin generierte Service Worker reagiert auf die Message `SKIP_WAITING` mit einem – öhm ja - `skipWaiting`. 😊

Verwende daher, um die Funktionalität zu implementieren unter anderem:

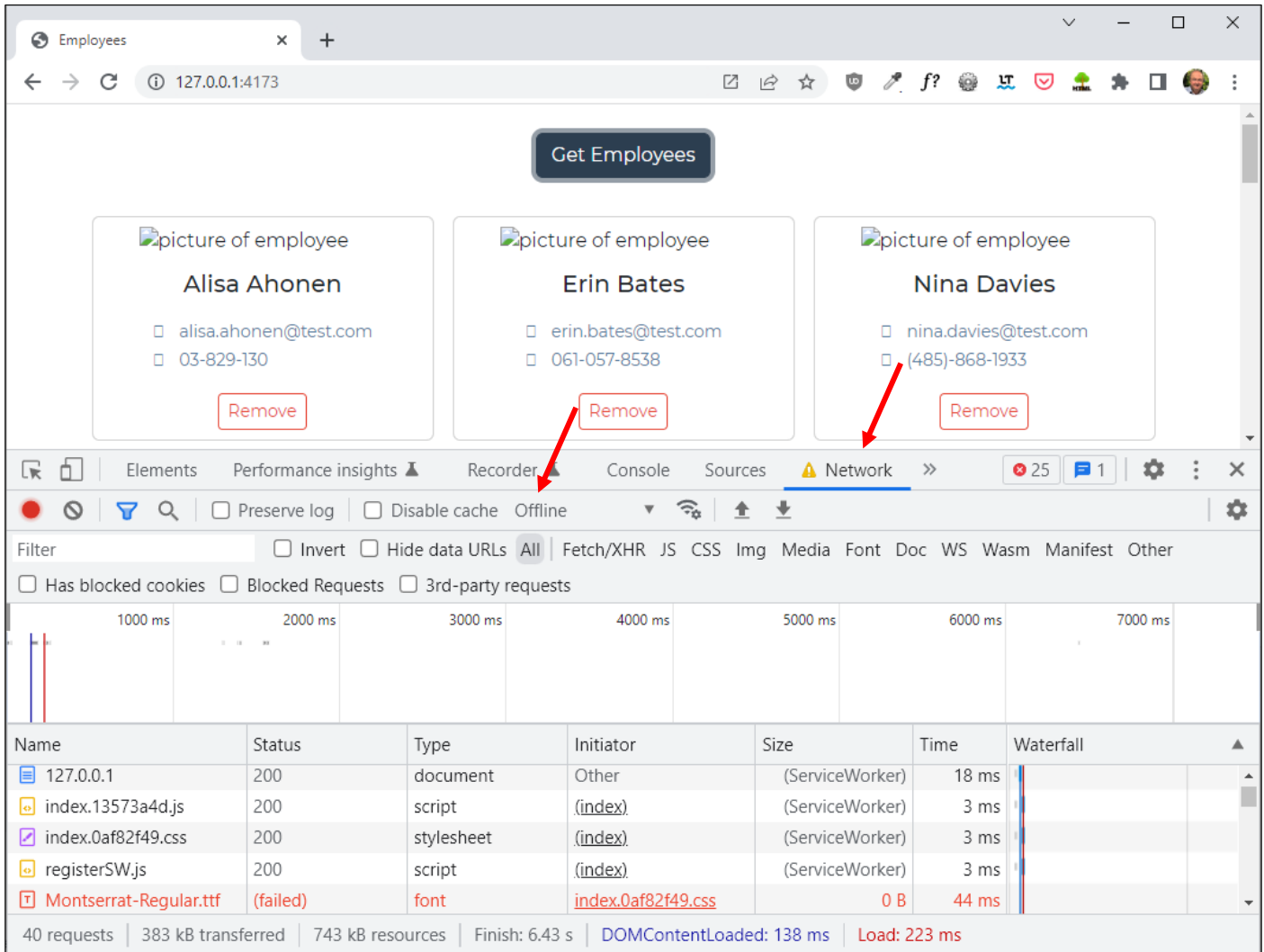
```
const registration = await navigator.serviceWorker.getRegistration();
if (registration) registration.waiting?.postMessage({ type: 'SKIP_WAITING' });
window.location.reload();
```

Aufgabe 6: Erweitere die Funktionalität, sodass der Benutzer mit einem Button den neuen Service Worker installieren kann und die Seite neu lädt!

Robert Baumgartner

Zum Abschluss wollen wir noch die Bilder vom **/public** Verzeichnis als Precache speichern. Erinnerung: sie standardmäßig nicht dabei sind.

Gehe im Netzwerk Tab auf **offline** und lade die Seite neu.



Du siehst: Es gibt keine Bilder und Icons!

Im Vite PWA Plugin können wir für die Workbox Muster definieren (Glob Patterns²), die angeben, was alles in den Precache kommen soll. Die Glob Patterns werden von WorkBox verwendet, um Dateien im dist-Ordner abzugleichen.

Dazu kannst du das **workbox** property in **vite.config.js** angeben. Achtung, du musst aber alle Files angeben, die precached werden sollen. Das ist das Default Pattern:

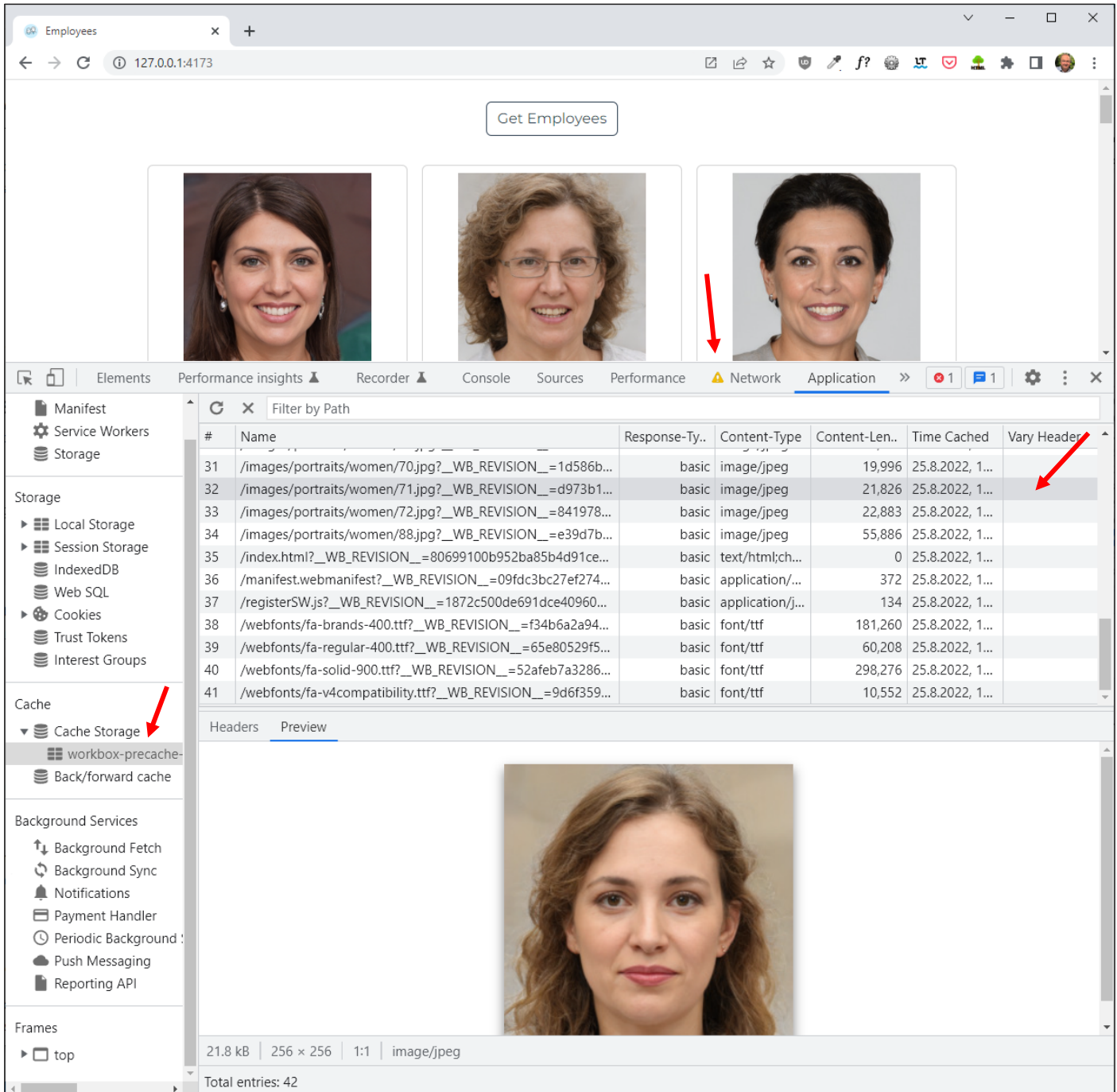
```
workbox: {
  globPatterns: ['**/*.js,css,html,ico'],
},
```

Aufgabe 7: Gib ein Glob Pattern an, dass auch Icons, Fonts und die Bilder precached!

² Siehe: <https://datacadamia.com/lang/regex/glob>

Robert Baumgartner

Überprüfe dein Ergebnis.



The screenshot shows a web browser window with the URL 127.0.0.1:4173. The page displays a 'Get Employees' button and three employee portraits. The Network tab is open, showing a list of resources. A red arrow points to the 'workbox-precache-' entry in the Cache section.

#	Name	Response-Ty..	Content-Type	Content-Len..	Time Cached	Vary Header
31	/images/portraits/women/70.jpg?__WB_REVISION__=1d586b...	basic	image/jpeg	19,996	25.8.2022, 1...	
32	/images/portraits/women/71.jpg?__WB_REVISION__=d973b1...	basic	image/jpeg	21,826	25.8.2022, 1...	
33	/images/portraits/women/72.jpg?__WB_REVISION__=841978...	basic	image/jpeg	22,883	25.8.2022, 1...	
34	/images/portraits/women/88.jpg?__WB_REVISION__=e39d7b...	basic	image/jpeg	55,886	25.8.2022, 1...	
35	/index.html?__WB_REVISION__=80699100b952ba85b4d91ce...	basic	text/html;ch...	0	25.8.2022, 1...	
36	/manifest.webmanifest?__WB_REVISION__=09fdc3bc27ef274...	basic	application/...	372	25.8.2022, 1...	
37	/registerSW.js?__WB_REVISION__=1872c500de691dce40960...	basic	application/j...	134	25.8.2022, 1...	
38	/webfonts/fa-brands-400.ttf?__WB_REVISION__=f34b6a2a94...	basic	font/ttf	181,260	25.8.2022, 1...	
39	/webfonts/fa-regular-400.ttf?__WB_REVISION__=65e80529f5...	basic	font/ttf	60,208	25.8.2022, 1...	
40	/webfonts/fa-solid-900.ttf?__WB_REVISION__=52afeb7a3286...	basic	font/ttf	298,276	25.8.2022, 1...	
41	/webfonts/fa-v4compatibility.ttf?__WB_REVISION__=9d6f359...	basic	font/ttf	10,552	25.8.2022, 1...	

The Cache section shows the following entries:

- workbox-precache-
- Back/forward cache

The Preview section shows a portrait of a woman with curly hair. The status bar indicates: 21.8 kB | 256 x 256 | 1:1 | image/jpeg. Total entries: 42.