

Caching von Dateien

Im Teil 2 hast du bereits erste Erfahrungen mit dem Service Worker (SW), vorwiegend mit dem Lebenszyklus, gemacht. Dabei wurde der SW automatisch durch den Buildprozess erzeugt.

In diesem Teil wollen wir selbst einen SW erstellen, um festzulegen, welche Daten gecached werden sollen.

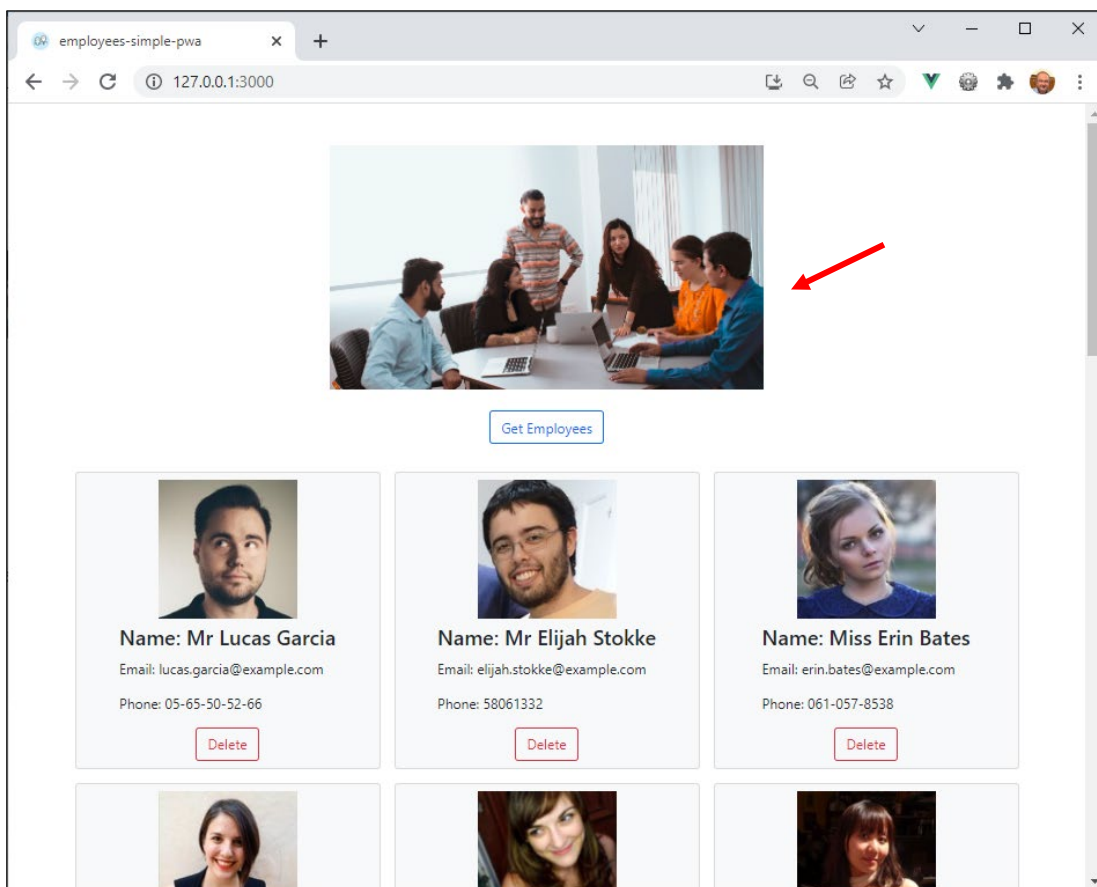
Aus der Natur des SWs ergeben sich einige Eigenschaften:

- Der SW kann er nicht auf **Local Storage (Session Storage)** zugreifen.
- Der SW läuft in einem isolierten Teil des Browsers und kann nicht auf das DOM zugreifen.
- Der SW verlangt aus Sicherheitsgründen:
 - eine HTTPS Verbindung (außer für **localhost**).
 - von der gleichen Seite geladen zu werden wie die App/Webseite
- Der SW kann Push Nachrichten empfangen, auch wenn die App nicht aktiv ist.
- Der SW hat seinen eigenen Lebenszyklus.
- Der SW reagiert auf Events.

Für diesen Teil arbeiten wir mit deinem Programm aus Teil 2 weiter.

Aufgabe 1: Führe die nachfolgenden Schritte durch.

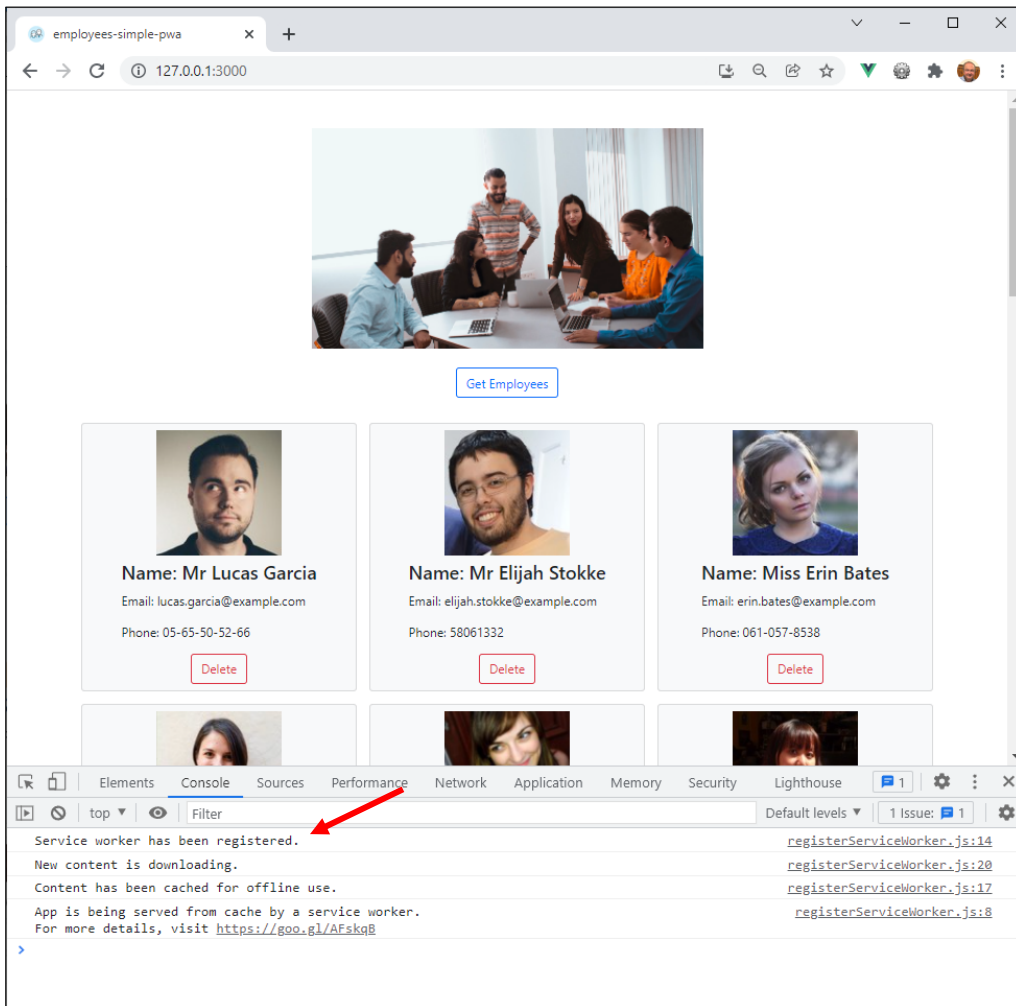
Füge das Bild **employees.jpg** zum Client hinzu (Ordner **/public** im Development-Verzeichnis des Clients).



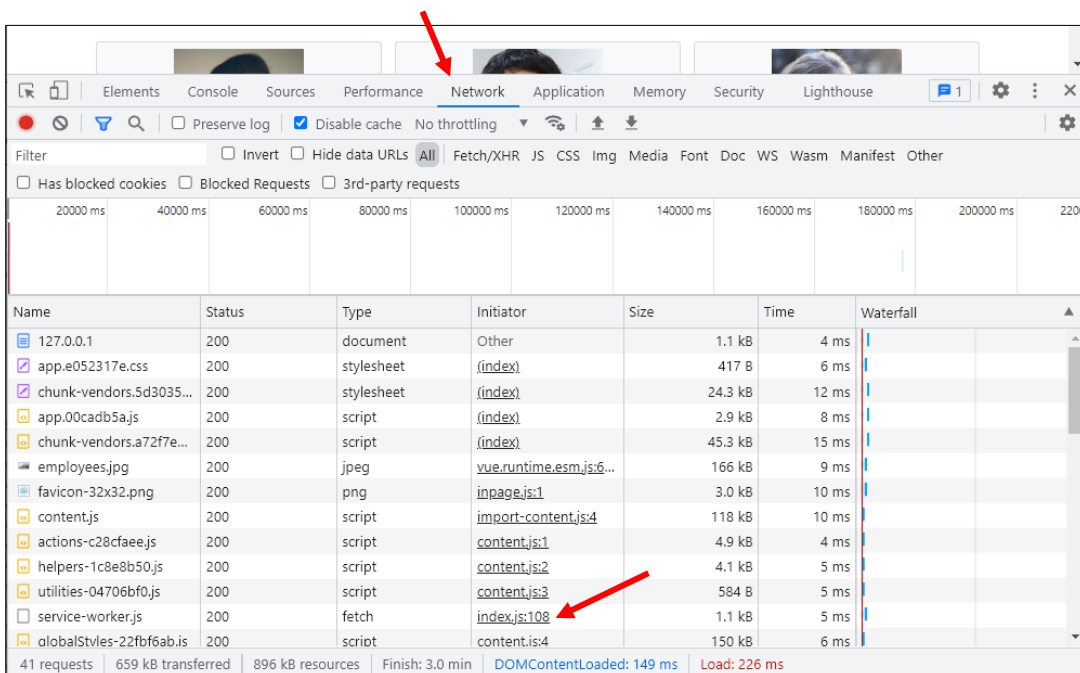
Bereinige das **/public** Verzeichnis (bis auf **/images**) am Server, starte den Server, baue dein Frontend (d. h. immer **npm run build**) und gehe mit dem Browser auf **http://127.0.0.1:3000**.

Robert Baumgartner

Checke die Console. Der Service Worker, den der Buildprozess gebaut hat, sollte aktiv sein (siehe nächste Seite).

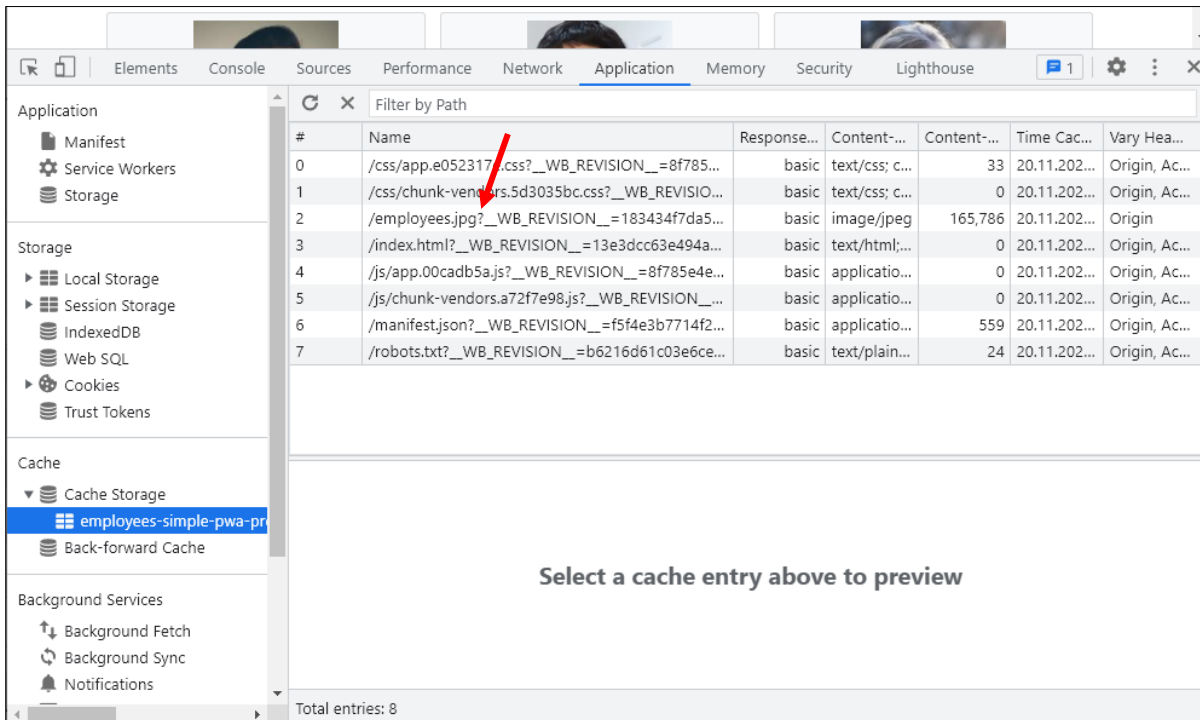


Gehe zum Netzwerk Tab. Dort siehst du, dass der Service Worker inklusive dem Rest deiner Webseite vom Server geladen wurde.

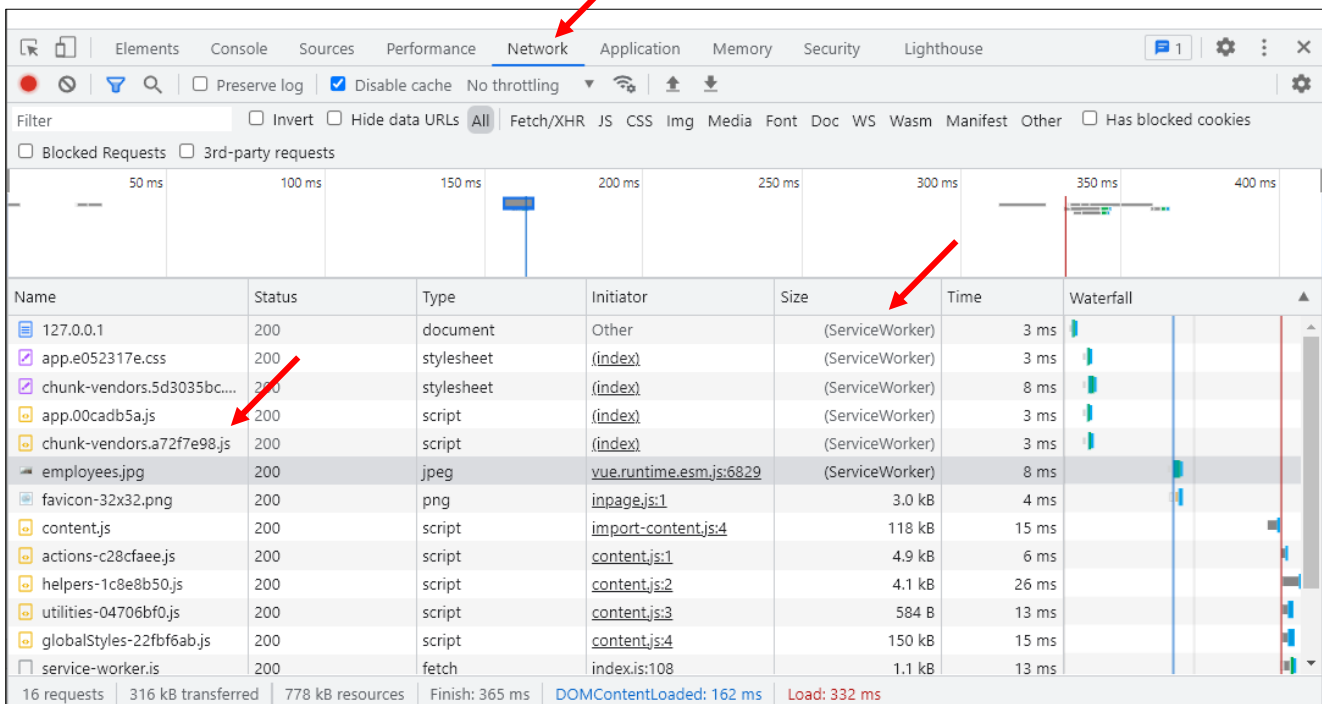


Robert Baumgartner

Prüfe den Cache deines Browsers. Die statischen Assets (HTML, CSS, JS,...) und auch das neue Bild **employees.jpg** wurde gecached (der Service Worker wurde nach dem Laden aktiviert und hat den Cache gefüllt).

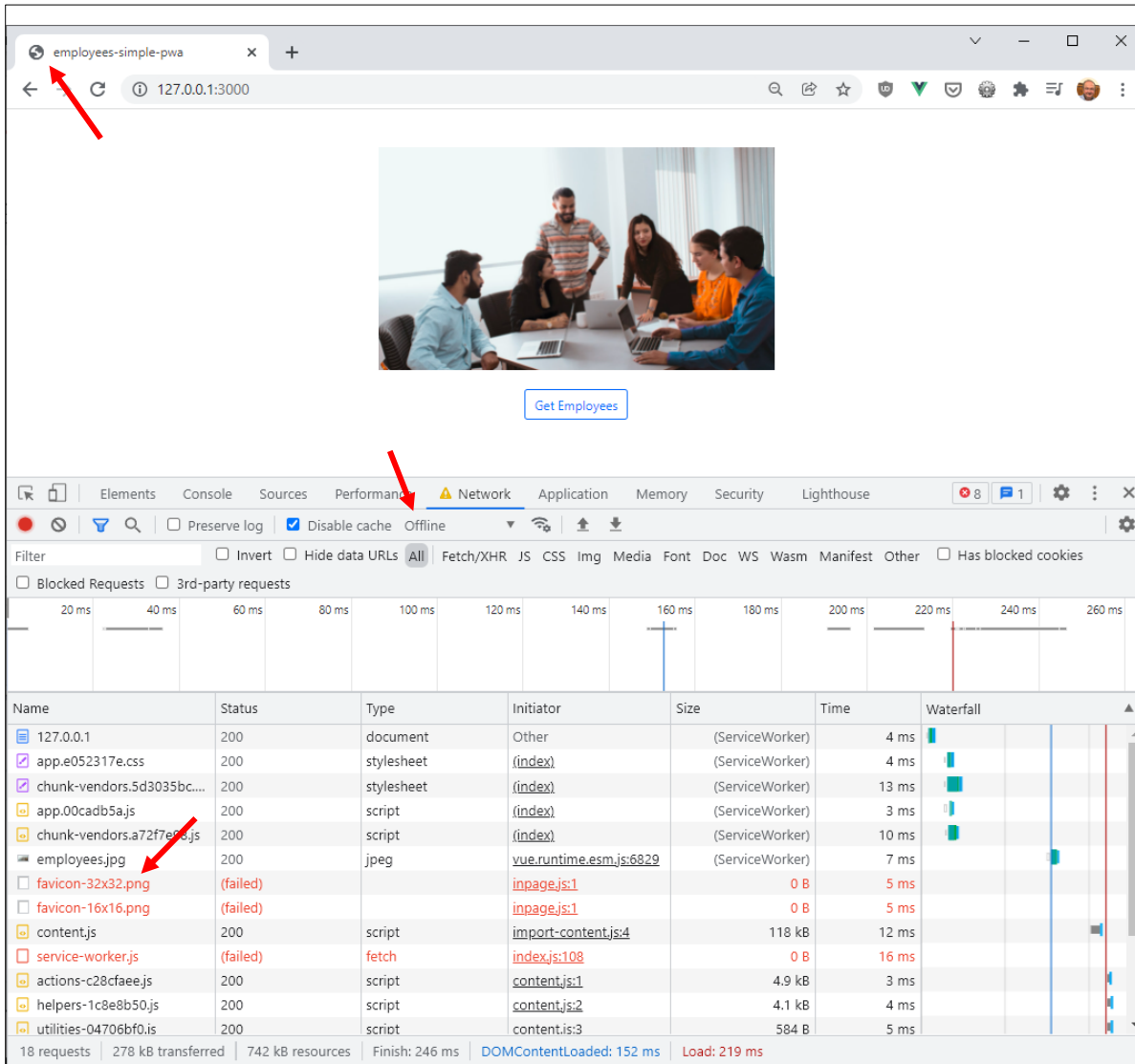


Ab nun werden die statischen Assets durch den Service Worker aus dem Cache geladen! Refreshe die Seite im Browser und beobachte den Netzwerk Tab! Als Quelle einiger Assets ist nun der Service Worker angegeben! Das bedeutet, wenn der Service Worker aktiv ist, werden einige Dateien aus dem Cache geladen. Egal ob du online oder offline bist.



Aufgabe 2: Um wieviel ist das Laden das JavaScript Bundles `chunk-vendors.a72f7e98.js` schneller geworden?

Gehe nun offline und refreshe die Seite! Der App Frame und das Bild werden, so wie vorher, aus den Cache via Service Worker geliefert. Einige Dateien wurden nicht gecached. Zum Beispiel das Fav Icon.



The screenshot shows a web browser window with the address bar displaying 'employees-simple-pwa' and the URL '127.0.0.1:3000'. The page content includes a photograph of five people in a meeting and a button labeled 'Get Employees'. The Chrome DevTools interface is open at the bottom, with the 'Network' tab selected. The network log shows various resources, including 'employees.jpg', which is marked as failed. A red arrow points to the 'employees.jpg' entry in the network log, and another red arrow points to the 'Network' tab in the DevTools toolbar.

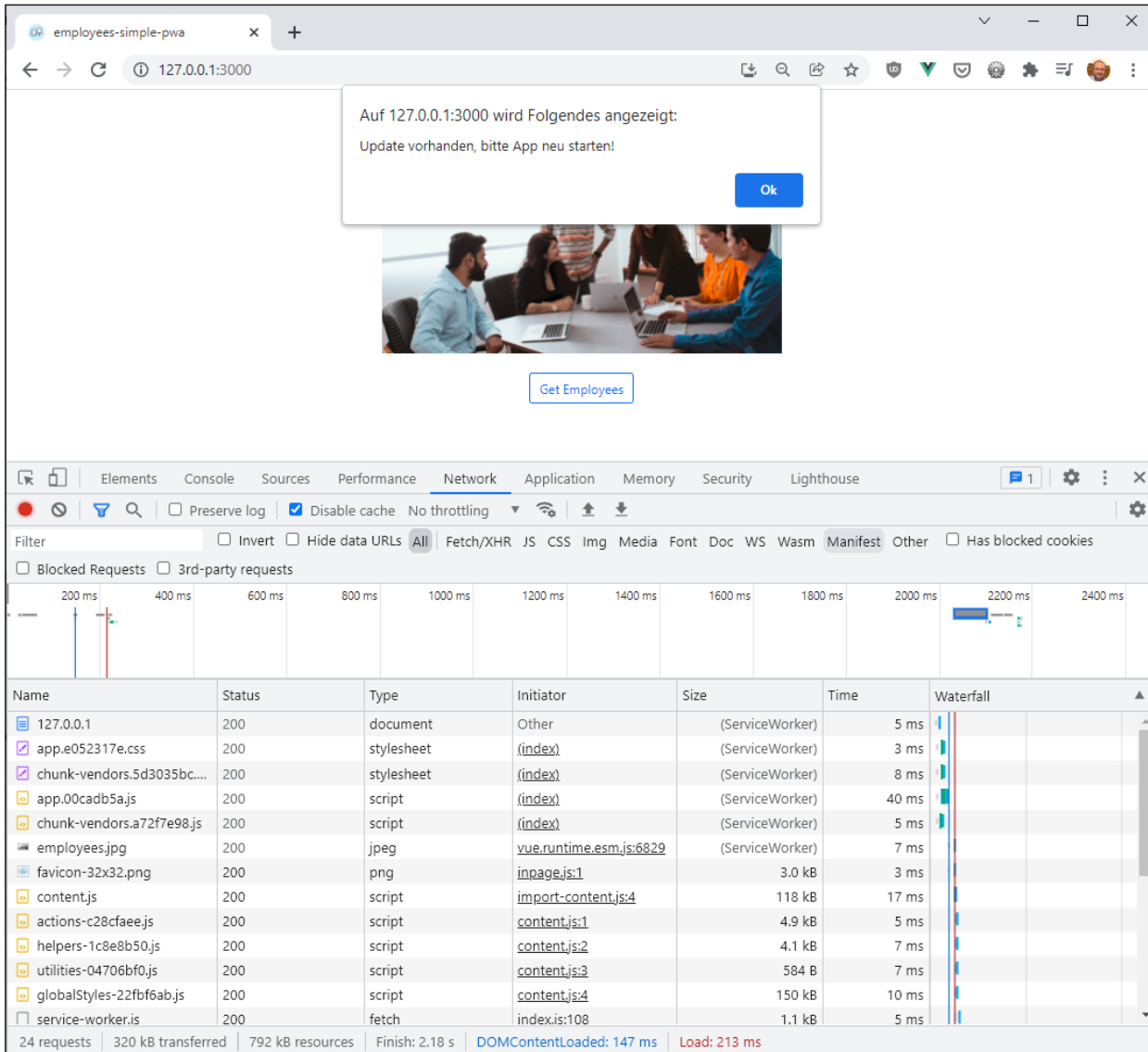
Name	Status	Type	Initiator	Size	Time	Waterfall
127.0.0.1	200	document	Other	(ServiceWorker)	4 ms	
app.e052317e.css	200	stylesheet	(index)	(ServiceWorker)	4 ms	
chunk-vendors.5d3035bc....	200	stylesheet	(index)	(ServiceWorker)	13 ms	
app.00cadb5a.js	200	script	(index)	(ServiceWorker)	3 ms	
chunk-vendors.a72f7e98.js	200	script	(index)	(ServiceWorker)	10 ms	
employees.jpg	200	jpeg	vue.runtime.esm.js:6829	(ServiceWorker)	7 ms	
favicon-32x32.png	(failed)		inpage.js:1	0 B	5 ms	
favicon-16x16.png	(failed)		inpage.js:1	0 B	5 ms	
content.js	200	script	import-content.js:4	118 kB	12 ms	
service-worker.js	(failed)	fetch	index.js:108	0 B	16 ms	
actions-c28cfaee.js	200	script	content.js:1	4.9 kB	3 ms	
helpers-1c8e8b50.js	200	script	content.js:2	4.1 kB	4 ms	
utilities-04706bf0.is	200	script	content.js:3	584 B	5 ms	

Summary: 18 requests | 278 kB transferred | 742 kB resources | Finish: 246 ms | DOMContentLoaded: 152 ms | Load: 219 ms

Gehe wieder online. Wie auch schon vorher, werden die statischen Assets aus dem Cache geladen.

Aufgabe 3: Füge eine Überschrift ein und baue den Build neu. Danach mache ein Refresh der Seite. Du solltest die von Teil 2 bekannte Update Meldung bekommen (siehe nächste Seite!)

Hier könnte Ihre Werbung stehen!



Auf 127.0.0.1:3000 wird Folgendes angezeigt:
Update vorhanden, bitte App neu starten!

Ok

Get Employees

Name	Status	Type	Initiator	Size	Time	Waterfall
127.0.0.1	200	document	Other	(ServiceWorker)	5 ms	
app.e052317e.css	200	stylesheet	(index)	(ServiceWorker)	3 ms	
chunk-vendors.5d3035bc...	200	stylesheet	(index)	(ServiceWorker)	8 ms	
app.00cadd5a.js	200	script	(index)	(ServiceWorker)	40 ms	
chunk-vendors.a72f7e98.js	200	script	(index)	(ServiceWorker)	5 ms	
employees.jpg	200	jpeg	vue.runtime.esm.js:6829	(ServiceWorker)	7 ms	
favicon-32x32.png	200	png	inpage.js:1	3.0 kB	3 ms	
content.js	200	script	import-content.js:4	118 kB	17 ms	
actions-c28cfaee.js	200	script	content.js:1	4.9 kB	5 ms	
helpers-1c8e8b50.js	200	script	content.js:2	4.1 kB	7 ms	
utilities-04706bf0.js	200	script	content.js:3	584 B	7 ms	
globalStyles-22fbf6ab.js	200	script	content.js:4	150 kB	10 ms	
service-worker.js	200	fetch	index.is:108	1.1 kB	5 ms	

24 requests | 320 kB transferred | 792 kB resources | Finish: 2.18 s | DOMContentLoaded: 147 ms | Load: 213 ms

Der aktive Service Worker hat die Webseite überprüft und festgestellt, dass sich etwas geändert hat. Wie macht er das?

Zunächst gibt es durch den Build einen neuen Service Worker. Dieser definiert, welches Pre-Cache Manifest gültig ist.

Was bedeutet Pre-Cache überhaupt? Wie du oben gesehen hast, ist eine Funktion des Service Workers das Speichern von Dateien im Browser Cache, wenn der Service Worker installiert wird. Dies wird oft als "Pre-Caching" bezeichnet, da die Dateien schon vor dem Einsatz des Service Workers zwischenspeichert werden (das können wir als Developer beeinflussen, siehe weiter unten).

service-workers.js:

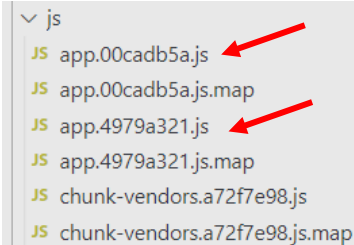
```
// ...
importScripts(
  "/precache-manifest.c3101afc4d92a3acc374faa9b2cd9d09.js"
);
// ...
```

Der Service Worker importiert also das Pre-Cache Manifest. Dadurch weiß er, welche Dateien zum Cachen bestimmt sind. Die Hashcodes sind die Fingerabdrücke der Dateien. Damit es zu keinen Fehlern kommt, sind alle Dateien mit einem Hashcode signiert.

Robert Baumgartner

Da du beim Bauen des Clients die Option `--no-clean` angegeben hast, solltest du sowohl die neue, wie auch die alte Version deines app Bundles am Server haben. Sie unterscheiden sich durch den Hashcode.

Vergleiche die beiden Hashes vor der Änderung durch die Überschrift und nach Einfügen der Überschrift. Geändert hat sich das Bundle `app.hashcode.js`.



```
js
├── app.00cadb5a.js
├── app.00cadb5a.js.map
├── app.4979a321.js
├── app.4979a321.js.map
├── chunk-vendors.a72f7e98.js
└── chunk-vendors.a72f7e98.js.map
```

Meine Hashes:

Alt: `precache-manifest.64cf7f1e484f06bd762c1c34f4bd176f.js`:

```
{
  "revision": "8f785e4ebc1eadfde026",
  "url": "/js/app.00cadb5a.js"
},
```

Neu: `precache-manifest.c3101afc4d92a3acc374faa9b2cd9d09.js`

```
{
  "revision": "5d6d356a99b77ef197ee",
  "url": "/js/app.4979a321.js"
},
```

Aufgabe 4: Wie lauten deine Pre-Cache Manifest Hashes von **app** vor und nach dem Hinzufügen der Überschrift?

Mit der Zeit sammeln sich die alten Versionen der Dateien im **/public** Ordner der Servers an. Das ist für das Testen ok, aber die alten Versionen sollten beim Deployment gelöscht werden (oder du erstellst einen eigenen Server mit einem zweiten **/public** Verzeichnis).

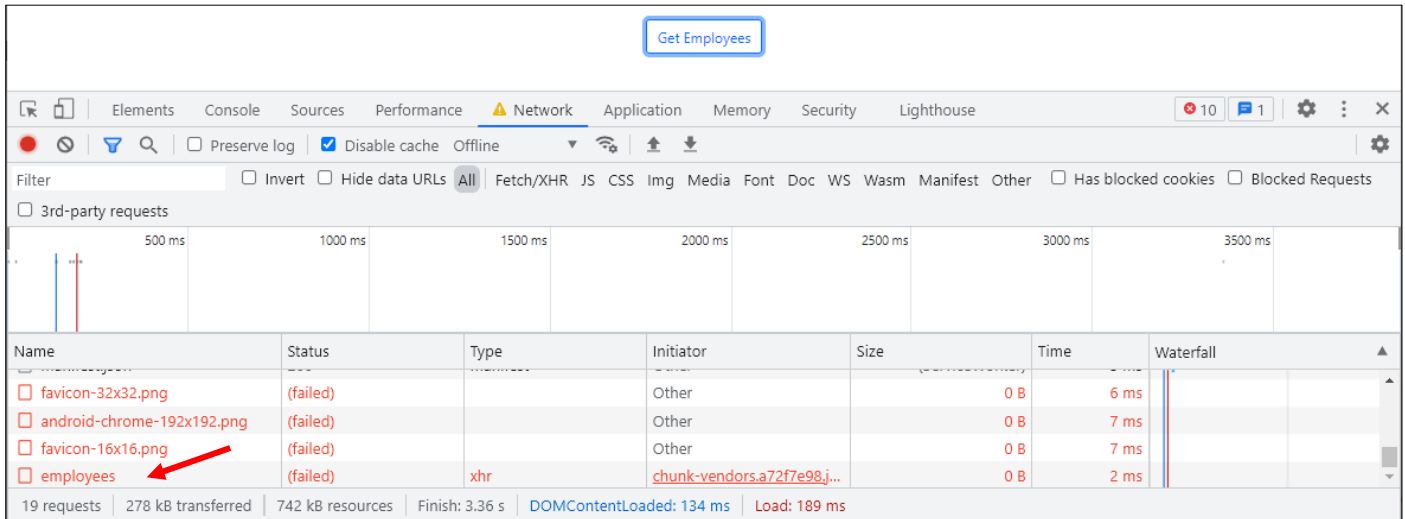
Nun zu den **.map** Dateien: Das sind Source Maps und dienen zum Debuggen. Sie werden sinnvollerweise standardmäßig nicht gecached. Wenn du sie nicht mehr benötigst, schalte mit folgender Option in **vue.config.js** das Erstellen der Source Maps aus. Nachteil: Dann kannst du den Programmcode im Browser nicht mehr lesen!



```
module.exports = {
  outputDir: path.resolve(__dirname, '../server/public'),
  productionSourceMap: false,
  pwa: {
    manifestOptions: {
      name: 'PWA Demo Employees',
      short_name: 'PWA Demo',
      theme_color: '#963484',
      background_color: '#28c2ff',
    },
  },
};
```

Forciertes Cachen von Dateien

Was ist mit den Daten (employees), die durch den Axios Call geladen werden? Werden auch die gecached? Die Antwort ist nein. Erstens kommen sie von einer anderen Route und zweitens werden dynamische JSON Calls standardmäßig nie gecached. Das kannst du überprüfen, wenn du offline gehst und auf den Button klickst.



The screenshot shows the Chrome DevTools Network tab with the 'Disable cache' checkbox checked. The 'employees' request is highlighted with a red arrow. The table below summarizes the visible requests:

Name	Status	Type	Initiator	Size	Time	Waterfall
favicon-32x32.png	(failed)		Other	0 B	6 ms	
android-chrome-192x192.png	(failed)		Other	0 B	7 ms	
favicon-16x16.png	(failed)		Other	0 B	7 ms	
employees	(failed)	xhr	chunk-vendors.a72f7e98.j...	0 B	2 ms	

Summary: 19 requests, 278 kB transferred, 742 kB resources, Finish: 3.36 s, DOMContentLoaded: 134 ms, Load: 189 ms.

Der Klick auf den Button triggert den Axios Call und fügt unten den Netzwerk Error des XHR Calls hinzu. Um auch diese Daten zu cachen, müssen wir selbst einen Service Worker erstellen.

Das Vue PWA Plugin verwendet den Google Baukasten für Service Worker: **Workbox**. Siehe: <https://developers.google.com/web/tools/workbox/>

Standardmäßig cached Workbox einige statische Dateien nicht. Deshalb fehlen sie auch in der obigen Ausgabe. Schau mal in den Code des PWA Plugins `employees-simple-pwa/node_modules/@vue/cli-plugin-pwa/index.js`. Da sind u. a. die Source Maps und die Icons vom `img` Verzeichnis ausgenommen!

```
const defaultOptions = {
  exclude: [
    /\.map$/,
    /img\/icons\/\//,
    /favicon\.ico$/,
    /^manifest.*\.js?$/
  ]
}
```

Das macht Sinn, denn sie werden beim ersten Download auf das Smartphone geladen und sind dann dort installiert. Geht die App offline, sind sie noch immer vorhanden. Auch das **favicon** wird in der Regel nicht benötigt, da es ja nicht angezeigt wird.

Damit Vue beim Buildprozess deinen Service Worker findet, musst du den `vue.config.js` File erweitern.


```
module.exports = {
  outputDir: path.resolve(__dirname, '../server/public'),
  pwa: {
    workboxPluginMode: 'InjectManifest',
    workboxOptions: {
      swSrc: 'public/service-worker.js',
    },
    manifestOptions: {
      name: 'PWA Demo Employees',
      short_name: 'PWA Demo',
      theme_color: '#963484',
      background_color: '#28c2ff',
    },
  },
};
```

Durch die Angabe von `swSrc` ist klar, wo der **service-workers.js** angelegt werden muss.

Der Mode `injectManifest` erlaubt mehr Kontrolle über den Service Worker. Dieser Mode wird auch standardmäßig vom PWA-Plugin verwendet und erzeugt, wie wir gesehen haben, beim Build Prozess im `/dist` Ordner eine eigene Datei mit allen Urls, die gecached werden. Eben das Pre-Cache Manifest.

Aufgabe 5: Erzeuge in `/public` (Client) eine Datei mit dem Namen **service-worker.js** und gib folgenden Code ein.

```
/* global workbox */

if (workbox) {
  console.log(`Workbox is loaded`);

  workbox.precaching.precacheAndRoute(self.__precacheManifest);
} else {
  console.log(`Workbox didn't load`);
}
```

Erstelle von deinem Client einen Build im `/public` Verzeichnis am Server..

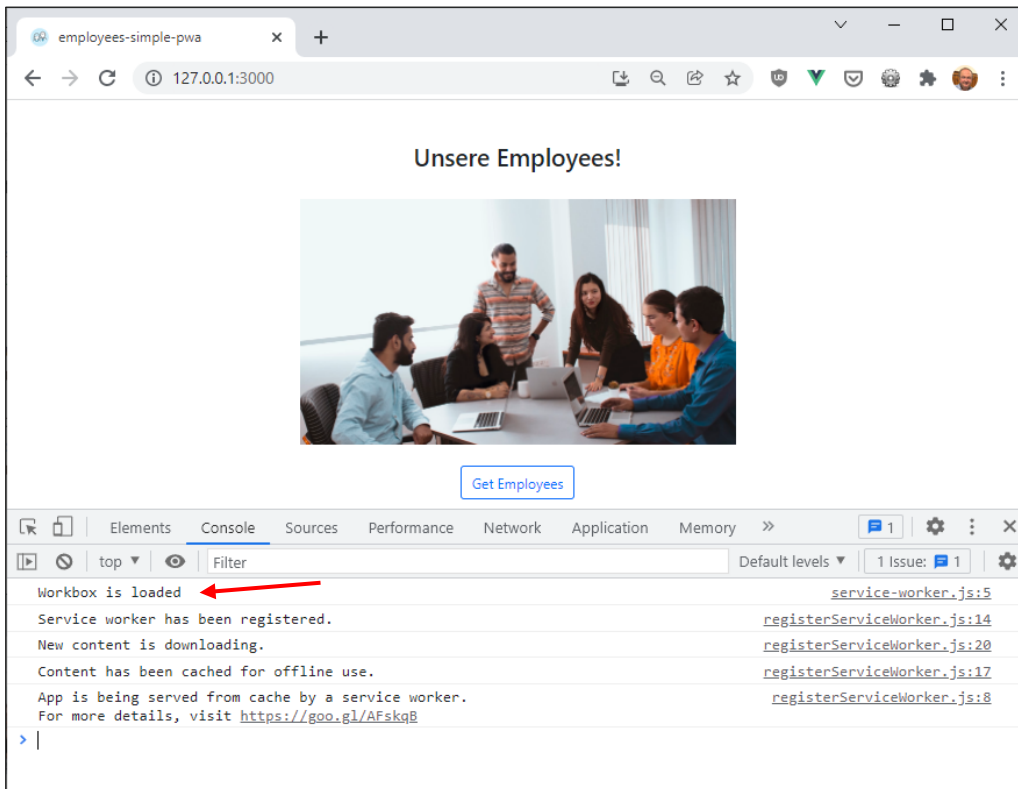
Schau dir den finalen Service Worker am Server an.

automatisch injected



Gehe auf <http://127.0.0.1:3000>

Lösche wenn nötig, den Cache im Browser!



Der eigene Service Worker ist also aktiv, der Application Frame wird standardmäßig injected. Nun können wir uns überlegen wie wir die Daten, die vom Axios Call geladen werden, cachen. Dazu bietet Workbox eine fertige Implementierung von verschiedenen Caching-Strategien an. Eine Caching-Strategie ist ein Muster, das bestimmt, wie ein Service Worker nach dem Empfang eines Fetch-Ereignisses (i. e. Axios Call) eine Antwort erzeugt. Erinnerung, der Service Worker (wenn aktiv) ist jeder Netzwerkanfrage deiner App zwischengeschaltet.

Lies dazu: <https://developers.google.com/web/tools/workbox/modules/workbox-strategies>

Zusammenfassung:

- **Cache first:** Alle Anfragen werden vom Service Worker aus dem Cache beantwortet. Ist er dazu nicht in der Lage, leitet der Service Worker die Anfrage an das Netzwerk (den Server) weiter. Die Antwort wird gecached.
- **Cache only:** Funktioniert wie bei Cache First. Kann der Cache keine Antwort geben, passiert allerdings nichts weiter. Wird eher selten benutzt.
- **Network first:** Zuerst schickt der Service Worker die Anfrage an das Netzwerk (den Server). Die Antwort wird gecached. Der Cache kommt erst zum Zug, wenn der Server nicht erreichbar ist.
- **Network only:** Wie Network first, allerdings kommt der Cache nie zum Einsatz.
- **Stale while revalidate:** Der Service Worker beantwortet die Anfrage aus dem Cache. Im Hintergrund vergleicht der Service Worker die Daten aus dem Cache mit den Daten vom Server und aktualisiert sie wenn nötig.

Aufgabe 6: Füge im Client in der Datei **service-worker.js** die Route **/employees** dem Service Worker hinzu. Wähle: **Network first**. Benenne gleichzeitig den Cache, indem die Daten abgespeichert werden. Schalte außerdem den Logger ein.

Lies dazu: <https://developers.google.com/web/tools/workbox/guides/route-requests>

Robert Baumgartner

```

if (workbox) {
  console.log(`Workbox is loaded`);

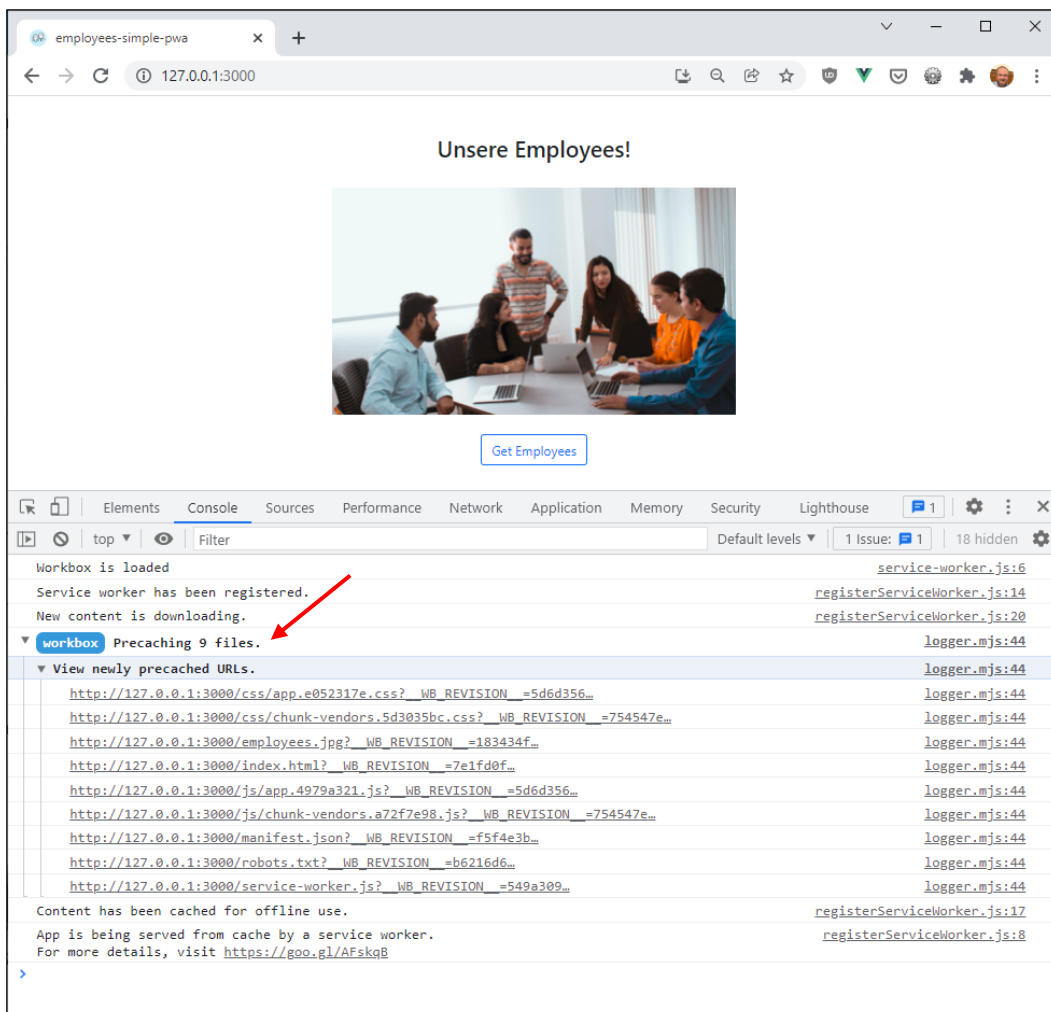
  workbox.setConfig({ debug: true });
  workbox.precaching.precacheAndRoute(self.__precacheManifest);
  workbox.routing.registerRoute(
    '/employees',
    new workbox.strategies.NetworkFirst({
      cacheName: 'roberts-cache',
    }),
  );
} else {
  console.log(`Workbox didn't load`);
}

```

Baue einen neuen Build, lösche die gecachten Daten und den Service Worker.

Lade die Seite. Checke Console (siehe nächste Seite).

Da wir für die Workbox Debug aktiviert haben, sehen wir, dass im Pre-Caching 9 Dateien heruntergeladen wurden. Unter anderem der neue Service Worker. Da kein Service Worker lief, wurde er aktiviert und übernimmt nach dem nächsten Reload je nach Caching Strategie das Ruder.

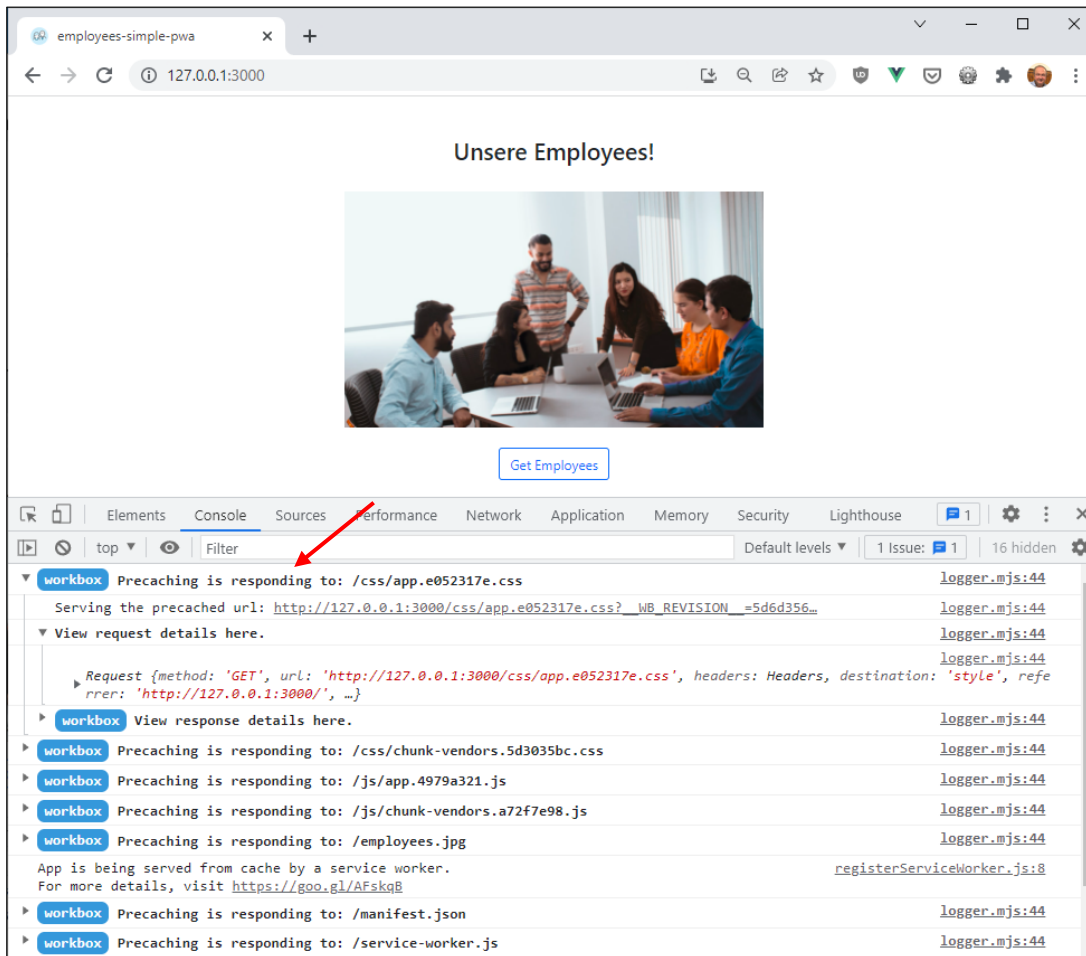


The screenshot shows a web browser window with the address bar displaying 'employees-simple-pwa' and the URL '127.0.0.1:3000'. The page content includes the heading 'Unsere Employees!', a photo of a group of people in an office, and a button labeled 'Get Employees'.

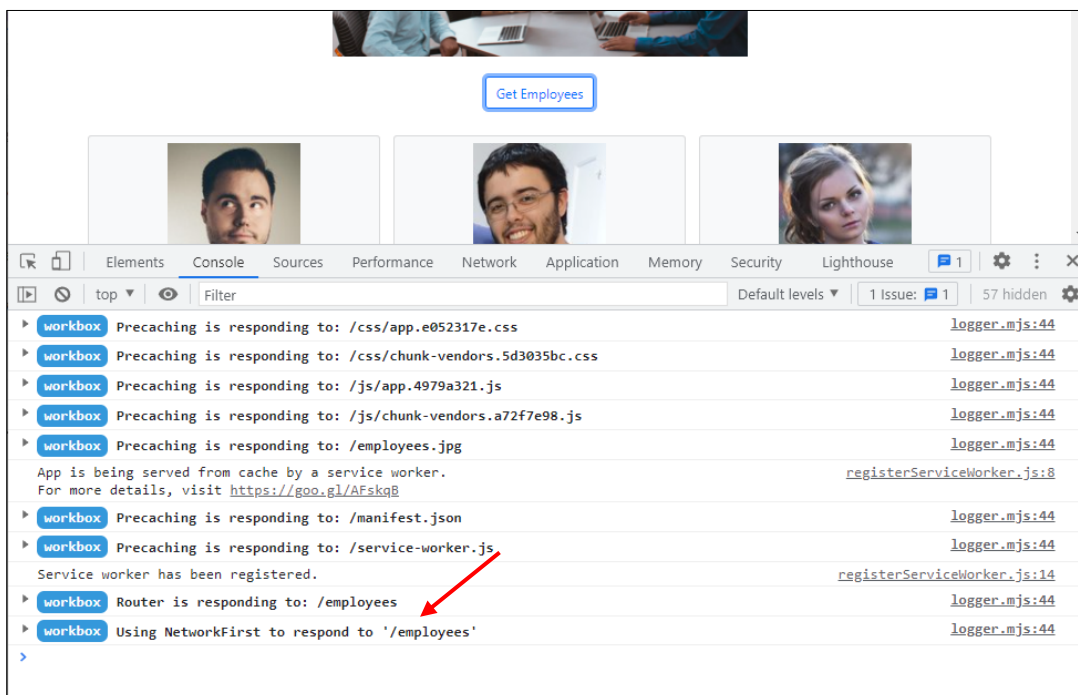
The browser's developer console is open, showing the following messages:

- Workbox is loaded
- Service worker has been registered.
- New content is downloading.
- workbox** Precaching 9 files.
- View newly precached URLs.**
 - http://127.0.0.1:3000/css/app.e052317e.css?_WB_REVISION_=5d6d356...
 - http://127.0.0.1:3000/css/chunk-vendors.5d3035bc.css?_WB_REVISION_=754547e...
 - http://127.0.0.1:3000/employees.jpg?_WB_REVISION_=183434f...
 - http://127.0.0.1:3000/index.html?_WB_REVISION_=7e1fd0f...
 - http://127.0.0.1:3000/js/app.4979a321.js?_WB_REVISION_=5d6d356...
 - http://127.0.0.1:3000/js/chunk-vendors.a72f7e98.js?_WB_REVISION_=754547e...
 - http://127.0.0.1:3000/manifest.json?_WB_REVISION_=f5f4e3b...
 - http://127.0.0.1:3000/robots.txt?_WB_REVISION_=b6216d6...
 - http://127.0.0.1:3000/service-worker.js?_WB_REVISION_=549a309...
- Content has been cached for offline use.
- App is being served from cache by a service worker.
- For more details, visit <https://goo.gl/AFskg8>

Refreshe die Seite. Der Service Worker übernimmt nun den Request. siehe Console:

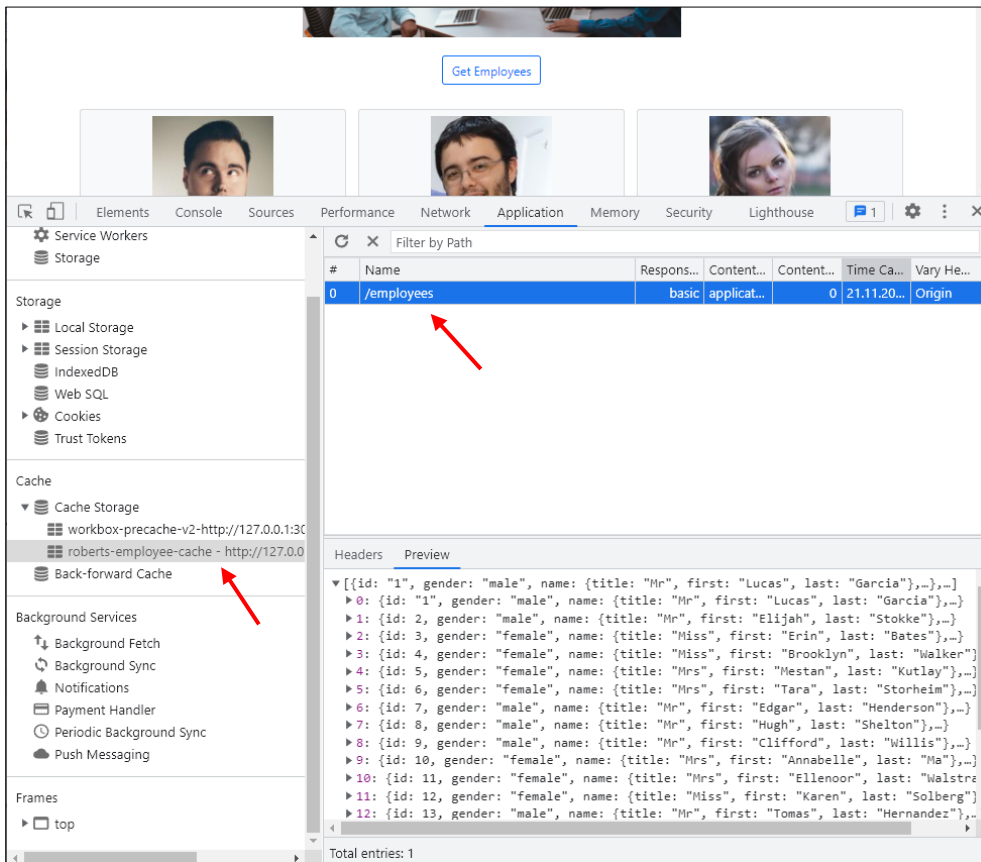


Klicke auf **Get Employees**, unsere Caching Strategie wird angewendet!



Robert Baumgartner

Der Cache ist gefüllt:



Get Employees

Cache Storage

- workbox-precache-v2-http://127.0.0.1:3000
- roberts-employee-cache - http://127.0.0.1:3000**
- Back-forward Cache

Background Services

- Background Fetch
- Background Sync
- Notifications
- Payment Handler
- Periodic Background Sync
- Push Messaging

Frames

- top

Filter by Path

#	Name	Respons...	Content...	Content...	Time Ca...	Vary He...
0	/employees	basic	applicat...	0	21.11.20...	Origin

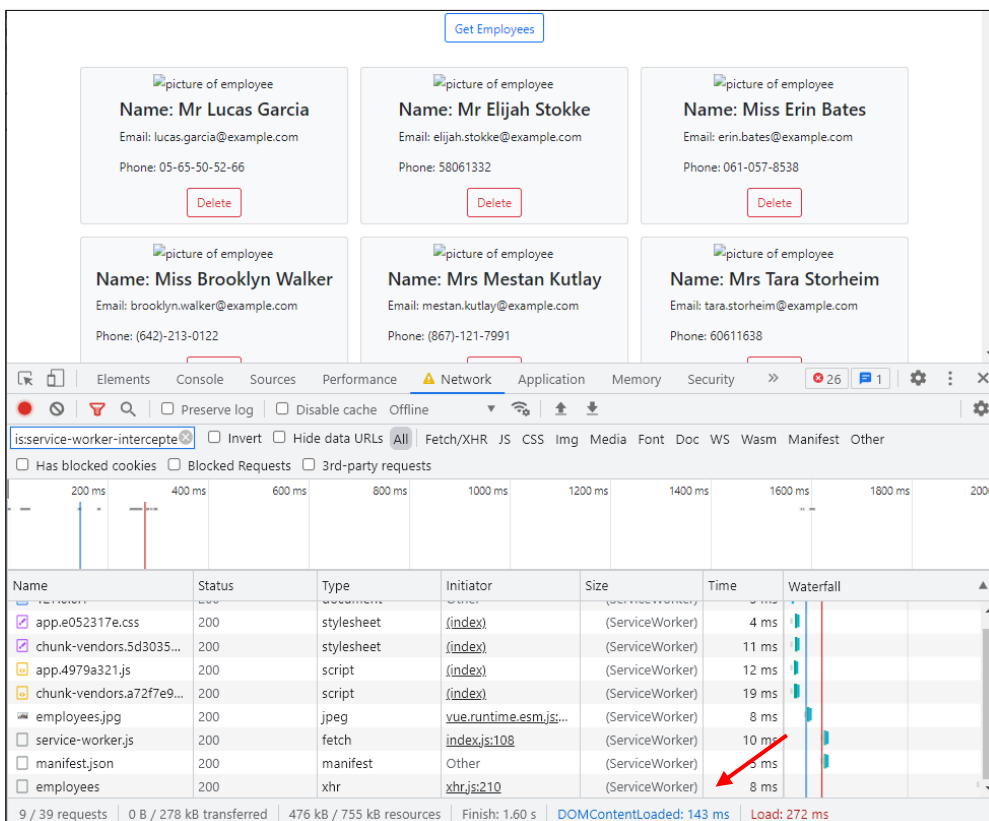
Headers Preview

```

[[{"id": "1", gender: "male", name: {title: "Mr", first: "Lucas", last: "Garcia"},...},
{"id": "2", gender: "male", name: {title: "Mr", first: "Elijah", last: "Stokke"},...},
{"id": "3", gender: "female", name: {title: "Miss", first: "Erin", last: "Bates"},...},
{"id": "4", gender: "female", name: {title: "Miss", first: "Brooklyn", last: "Walker"},...},
{"id": "5", gender: "female", name: {title: "Mrs", first: "Mestan", last: "Kutlay"},...},
{"id": "6", gender: "female", name: {title: "Mrs", first: "Tara", last: "Storheim"},...},
{"id": "7", gender: "male", name: {title: "Mr", first: "Edgar", last: "Henderson"},...},
{"id": "8", gender: "male", name: {title: "Mr", first: "Hugh", last: "Shelton"},...},
{"id": "9", gender: "male", name: {title: "Mr", first: "Clifford", last: "Willis"},...},
{"id": "10", gender: "female", name: {title: "Mrs", first: "Annabelle", last: "Ma"},...},
{"id": "11", gender: "female", name: {title: "Mrs", first: "Ellenoor", last: "Walstre"},...},
{"id": "12", gender: "female", name: {title: "Miss", first: "Karen", last: "Solberg"},...},
{"id": "13", gender: "male", name: {title: "Mr", first: "Tomas", last: "Hernandez"},...}]]
  
```

Total entries: 1

Gehe offline und überprüfe, ob die Daten aus dem Cache geladen werden:



Get Employees

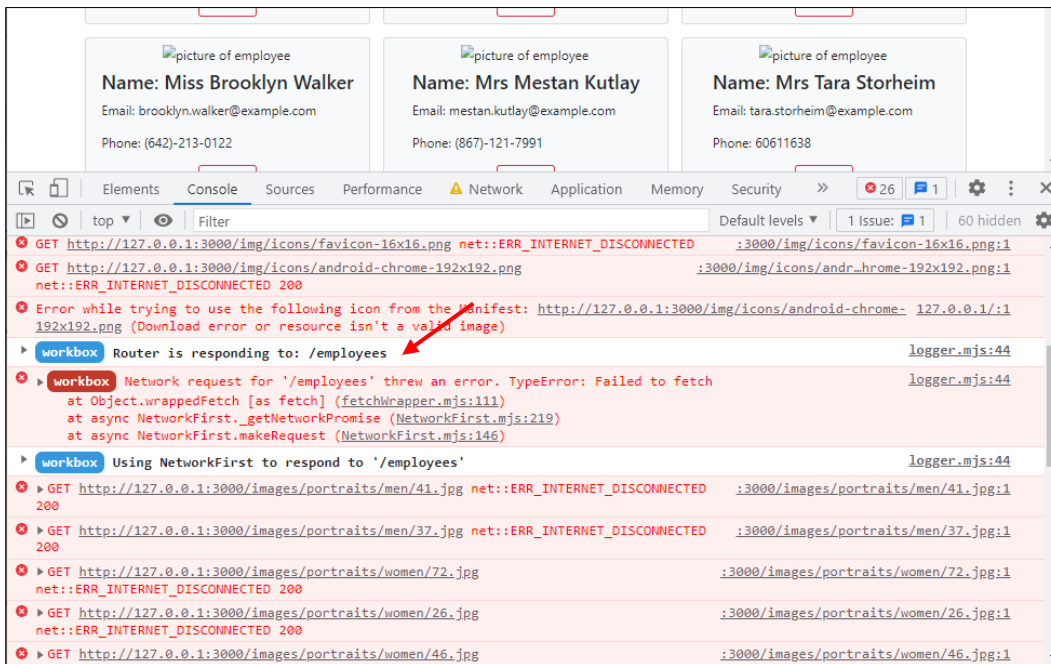
employees

9 / 39 requests | 0 B / 278 kB transferred | 476 kB / 755 kB resources | Finish: 1.60 s | DOMContentLoaded: 143 ms | Load: 272 ms

Name	Status	Type	Initiator	Size	Time	Waterfall
app.052317e.css	200	stylesheet	(index)	(ServiceWorker)	4 ms	
chunk-vendors.5d3035...	200	stylesheet	(index)	(ServiceWorker)	11 ms	
app.4979a321.js	200	script	(index)	(ServiceWorker)	12 ms	
chunk-vendors.a72f7e9...	200	script	(index)	(ServiceWorker)	19 ms	
employees.jpg	200	jpeg	vue.runtime.esm.js...	(ServiceWorker)	8 ms	
service-worker.js	200	fetch	index.js:108	(ServiceWorker)	10 ms	
manifest.json	200	manifest	Other	(ServiceWorker)	5 ms	
employees	200	xhr	xhr.js:210	(ServiceWorker)	8 ms	

Robert Baumgartner

Wenn du allerdings auf die Console siehst, stehen dort Fehler. Einige sind klar, weil die Dateien nicht gecached wurden, aber wieso der Fehler mit /employees? Die Daten werden doch aus dem Cache geladen und Workbox sagt auch: "responding to /employees".



Das ist anscheinend von den Developern so gewollt 🙄(ツ)🙄. Hier die Antwort von Google:

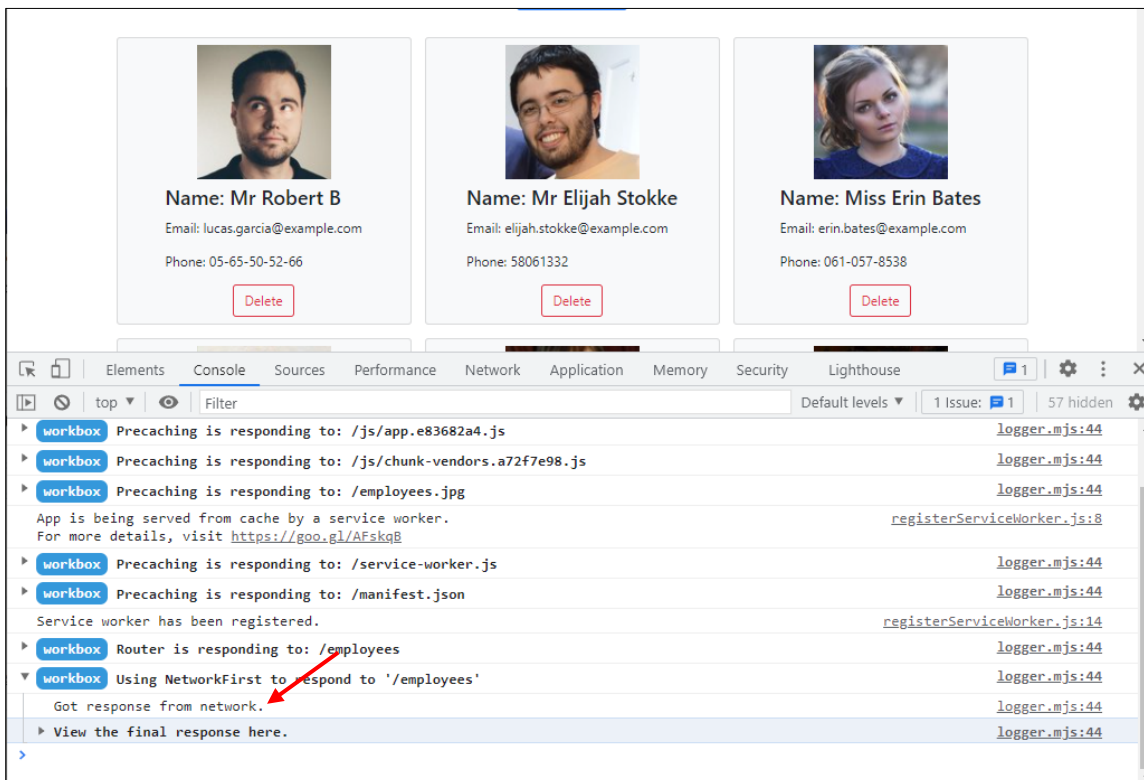
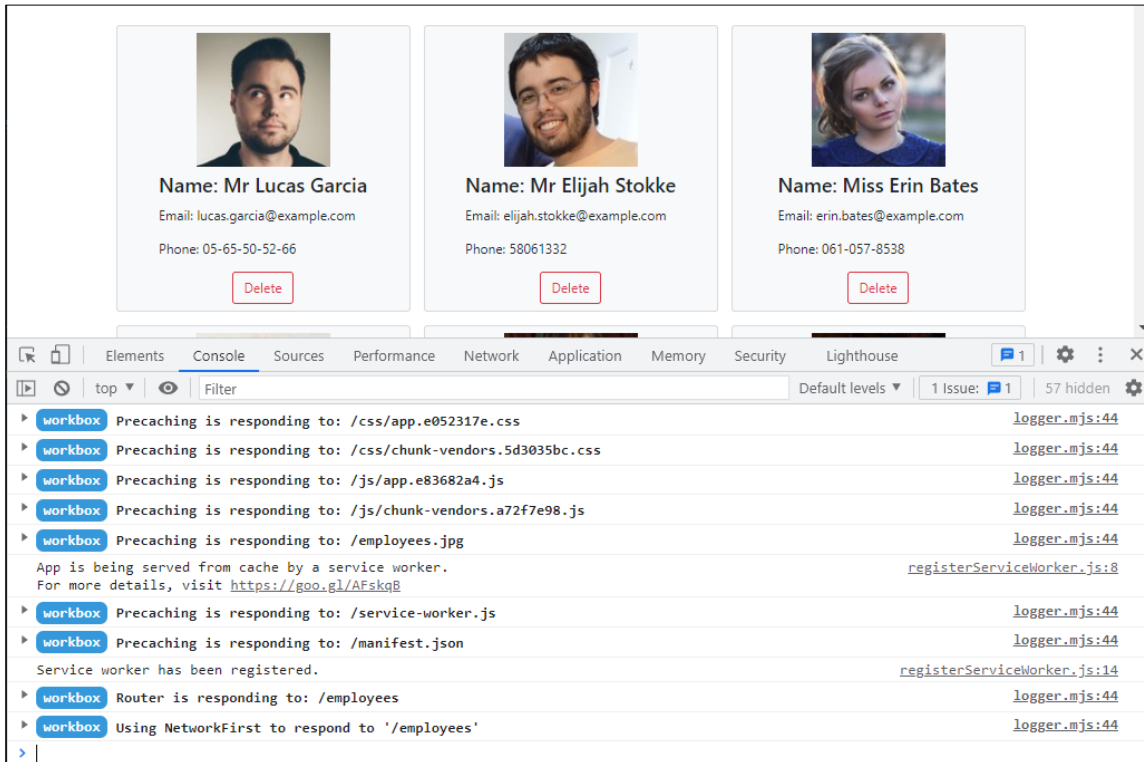


Aufgabe 7: Überprüfe nun die Strategie **Network first**. Bei laufendem Server und aktivem Service Worker hole die Daten vom Server. Mache einen Screenshot (siehe nächste Seite).

Ändere am Server im File **employees.js** den Namen von **Lucas Garcia** auf **deinen Namen**! Restarte den Server.

Refreshe die Seite und mache eine Screenshot (siehe nächste Seite).

Robert Baumgartner

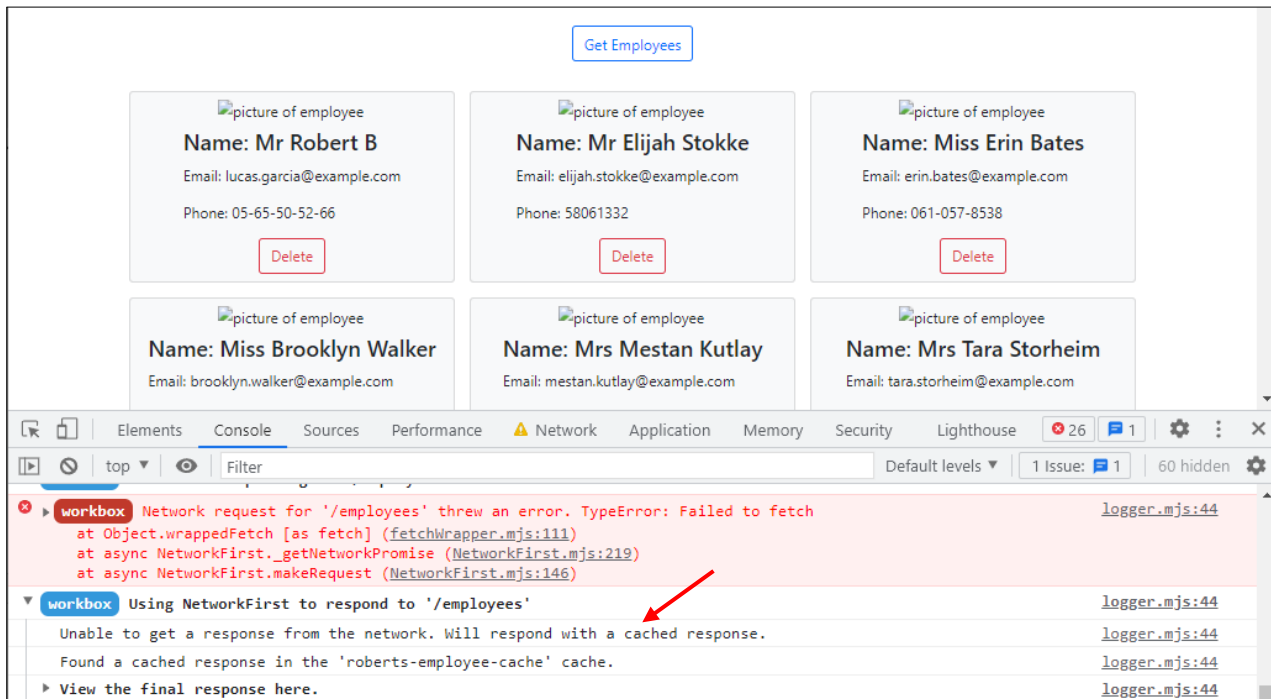


Du bekommst die Meldung, dass die Daten vom Netzwerk geladen wurden.

Gehe nun offline.

Refreshe die Seite und mache eine Screenshot (siehe nächste Seite).

Robert Baumgartner



Du bekommst die Meldung, dass die Daten vom Cache geladen wurden.

Ändere den Namen am Server wieder auf **Lucas Garcia**.

Aufgabe 8: Überprüfe nun selbstständig und analog zu dem vorigem Beispiel mit Screenshots die Strategie **Cache first**. Beschreibe deine Beobachtungen.

Nun zu den Bildern.

Hier wähle am besten **StaleWhileRevalidate**.

Aufgabe 9: Hast du eine Idee wieso?

Füge folgende Route zum Service Worker hinzu:

```
workbox.routing.registerRoute(
  new RegExp('/images/.*.jpg'),
  new workbox.strategies.StaleWhileRevalidate({
    cacheName: 'roberts-image-cache',
  }),
);
```


Aufgabe 10: Teste die neue Route online und offline!

Mache einen Screenshot.




employees-simple-pwa x +

127.0.0.1:3000

Unsere Employees!



Get Employees



Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
- Trust Tokens

Cache

Filter by Path

#	Name	Response...	Content...	Content-L...	Time Cac...	Vary Hea...
0	/images/portraits/men/20.jpg	basic	image/jpeg	4,726	22.11.202...	Origin
1	/images/portraits/men/22.jpg	basic	image/jpeg	4,752	22.11.202...	Origin
2	/images/portraits/men/27.jpg	basic	image/jpeg	4,051	22.11.202...	Origin
3	/images/portraits/men/37.jpg	basic	image/jpeg	5,084	22.11.202...	Origin
4	/images/portraits/men/41.jpg	basic	image/jpeg	4,254	22.11.202...	Origin
5	/images/portraits/men/59.jpg	basic	image/jpeg	2,799	22.11.202...	Origin
6	/images/portraits/women/0.jpg	basic	image/jpeg	4,857	22.11.202...	Origin
7	/images/portraits/women/26.jpg	basic	image/jpeg	4,680	22.11.202...	Origin
8	/images/portraits/women/36.jpg	basic	image/jpeg	5,093	22.11.202...	Origin
9	/images/portraits/women/37.jpg	basic	image/jpeg	5,554	22.11.202...	Origin
10	/images/portraits/women/46.jpg	basic	image/jpeg	5,782	22.11.202...	Origin

Total entries: 19

Console

Filter

workbox Router is responding to: /images/portraits/women/88.jpg

workbox Router is responding to: /images/portraits/men/22.jpg

workbox Router is responding to: /images/portraits/women/59.jpg

workbox Router is responding to: /images/portraits/women/0.jpg

workbox Using StaleWhileRevalidate to respond to '/images/portraits/men/41.jpg'

workbox Using StaleWhileRevalidate to respond to '/images/portraits/men/37.jpg'

workbox Using StaleWhileRevalidate to respond to '/images/portraits/women/72.jpg'