

Robert Baumgartner

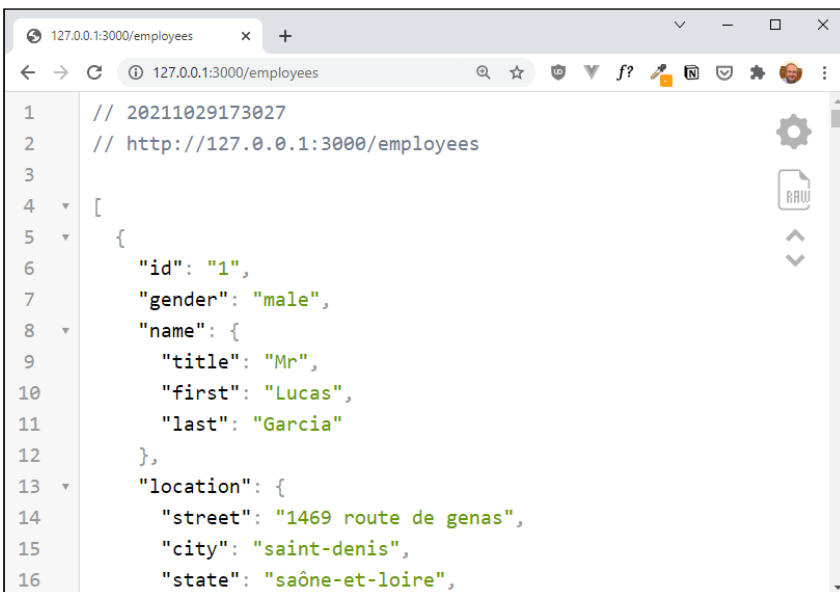
Erinnern wir uns nochmal an die **drei Voraussetzungen für eine PWA**: **manifest.json** Datei, **Service Worker** und eine **HTTPS Verbindung**. Wenn wir jedoch mit **localhost** arbeiten, können wir die PWA auch ohne HTTPS Verbindung testen. Wir werden die App allerdings nicht auf ein Smartphone laden können. Daher werden wir später unsere App Client+Server im Internet mit HTTPS hosten müssen.

Zunächst wollen wir uns aber den Service Worker genauer ansehen. Dafür reicht auch eine **localhost** Verbindung.

Nachdem du im ersten Teil mit einem weitgehend unveränderten Client gearbeitet hast, wird es Zeit den richtigen Client und damit die PWA zu erstellen.

**Aufgabe 1:** Entpacke **employees-simple-pwa START 2.zip**. Die Angabe besteht aus einem Server und einem Client.

Installiere die Module (Client und Server), starte den Server (**npm start**) und teste ihn.

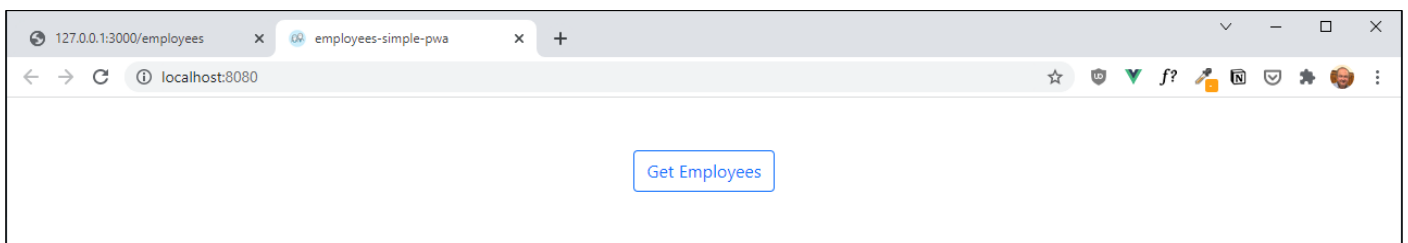


```
// 20211029173027
// http://127.0.0.1:3000/employees

[
  {
    "id": "1",
    "gender": "male",
    "name": {
      "title": "Mr",
      "first": "Lucas",
      "last": "Garcia"
    },
    "location": {
      "street": "1469 route de genas",
      "city": "saint-denis",
      "state": "saône-et-loire",

```

Sobald der Server läuft, starte den Client. Es wird nur der **Get Employees** Button angezeigt.



**Aufgabe 2:** Ergänze nun den Client, sodass die Employees vom Server angezeigt werden und der **Delete** Button (inkl. GUI Refresh) funktioniert.

Dazu musst du unter anderem Axios einbinden und Calls zum Server in den Methoden in **App.vue** programmieren.

Da wir später den Client sowohl Standalone (zum Testen) verwenden, als auch als Teil des Servers, sollte der Code tunlichst ohne Änderung in beiden Varianten funktionieren. Das geht aber schwer, wenn localhost hardcoded in **App.vue** steht! Daher lagern wir die Serveradresse in eine **.env** Datei aus.

Füge dem Client Root Verzeichnis zwei **.env** Dateien hinzu:

Robert Baumgartner

|                               |   |
|-------------------------------|---|
| <code>.env.development</code> | enthält die Definitionen für das Testen mit dem Developmentserver |
| <code>.env.production</code>  | enthält die Definitionen für die Production                       |

Im Zuge der Erstellung des Clients wird dann automatisch die richtige Environment Datei verwendet<sup>1</sup>! Im Code kannst du in gewohnter Weise mit `process.env.variable` darauf zugreifen.

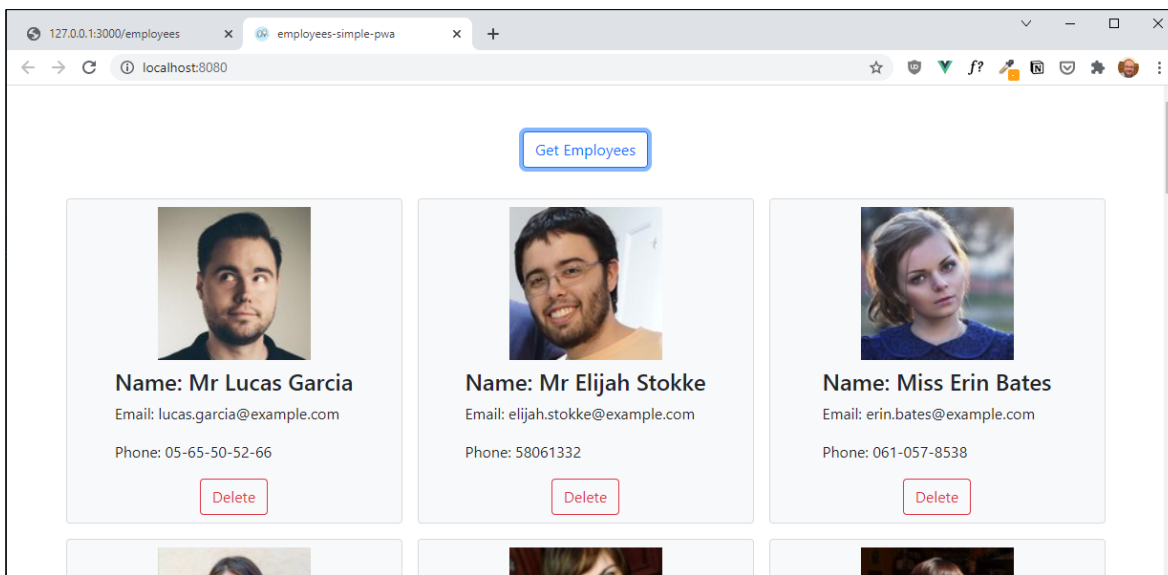
Beispiel:

```
VUE_APP_SERVER=http://localhost:3000
```

```
serverAddress: process.env.VUE_APP_SERVER
```

Beide Dateien `.env` enthalten zunächst die gleiche Server Adresse (wir ändern dann später die `.env.production` Datei!

So soll der Client aussehen (Daten kommen vom Server):



Nun wollen wir den Service Worker untersuchen. Da im Development Modus kein Service Worker vorhanden ist, benötigen wir den Buildprozess, der einen Service Worker erzeugt.

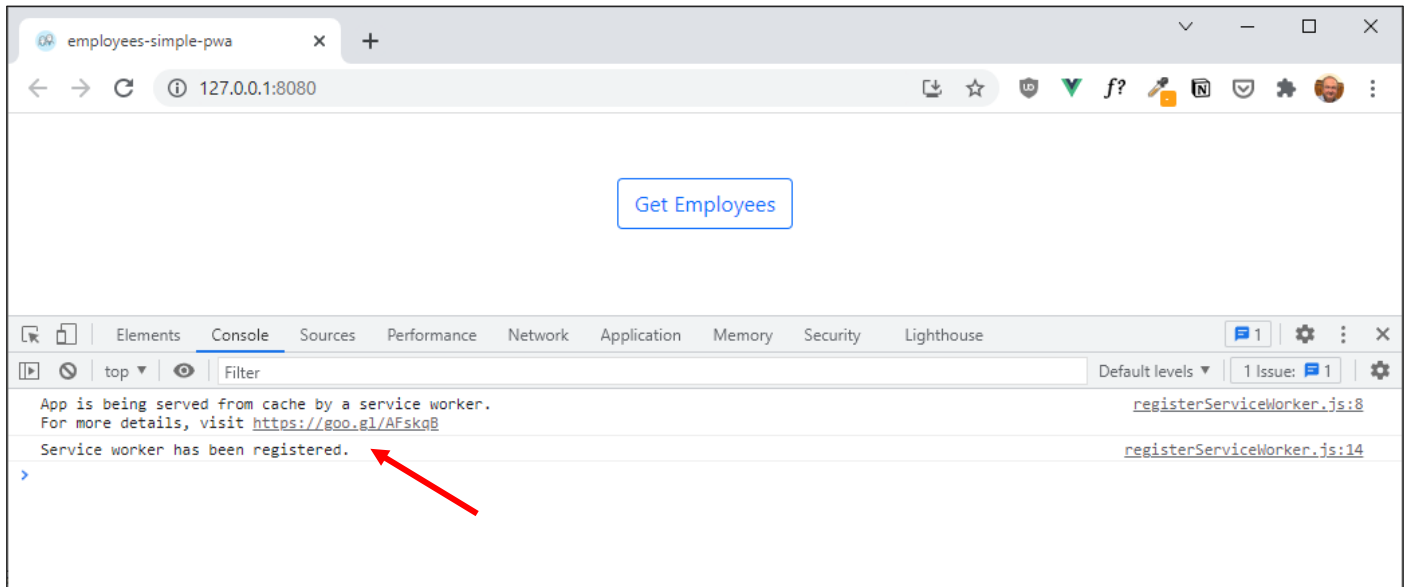
**Aufgabe 3:** Erstelle einen Build vom Client und starte den **http-server** mit dem **/dist** Verzeichnis.

An besten erstellst du ein Script in **package.json**:

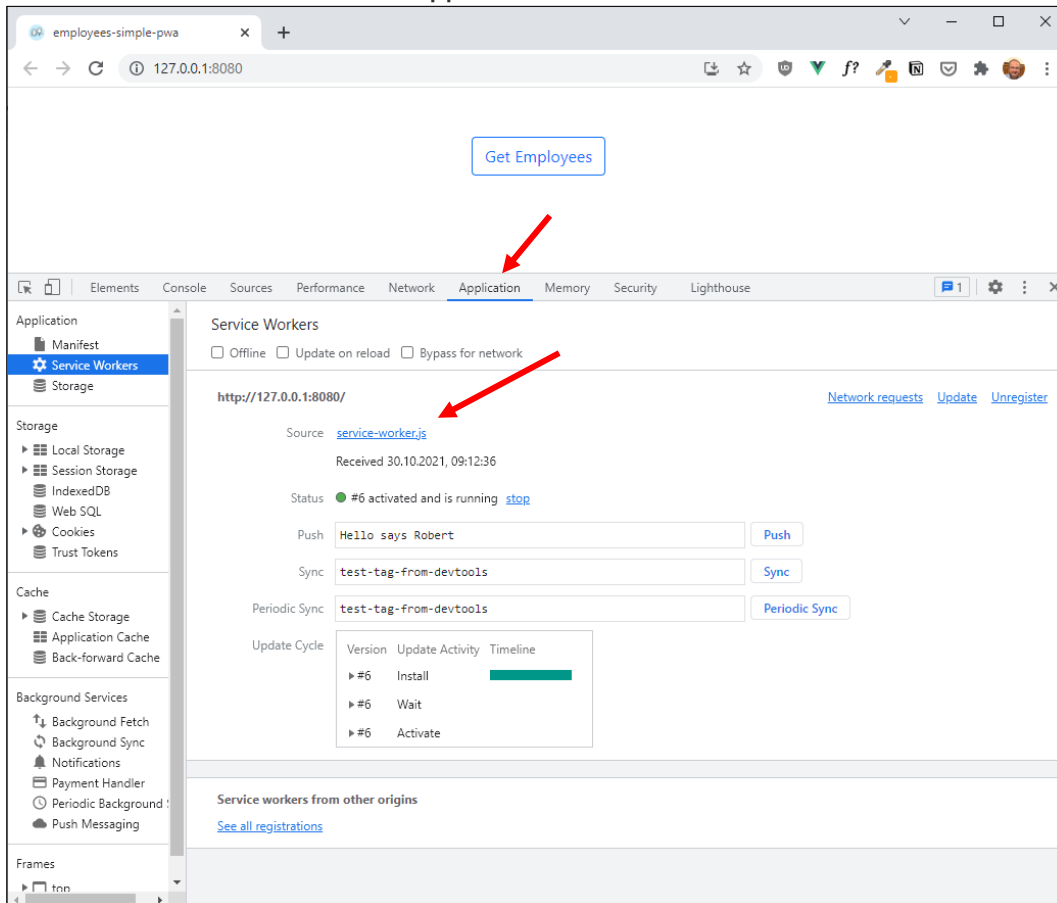
```
"scripts": {  
  "serve": "vue-cli-service serve",  
  "build": "vue-cli-service build",  
  "start": "http-server ./dist",  
  "lint": "vue-cli-service lint"  
},
```

Starte den Server, öffne den Browser und inspiere die Console. Der Service Worker läuft.

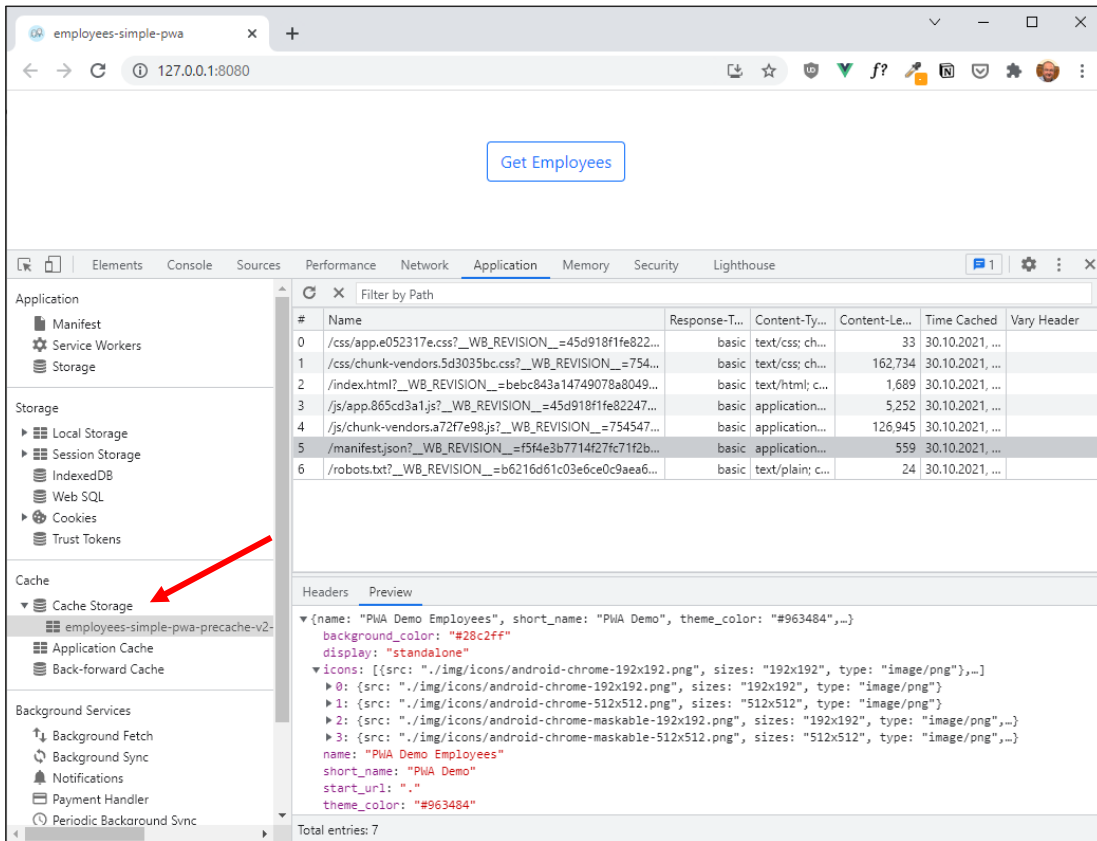
<sup>1</sup> Siehe: <https://cli.vuejs.org/guide/mode-and-env.html#modes>



Wechsle in den Dev Tools zum Tab **Application**.



Der Service Worker cached per Default die Basisfiles deiner App (js, css, etc.). Diese Basisfiles heißen **App Frame**.



The screenshot shows the Chrome DevTools Application tab. The left sidebar is expanded to 'Cache', and the 'Cache Storage' section is selected. A red arrow points to the entry 'employees-simple-pwa-precache-v2'. The main pane shows the details of this cache entry, including its name, short name, theme color, and a list of icons.

| # | Name  | Response-T... | Content-Ty...    | Content-Le... | Time Cached     | Vary Header |
|---|---|---------------|------------------|---------------|-----------------|-------------|
| 0 | /css/app.e052317e.css?_WB_REVISION_=45d918f1fe822...  | basic         | text/css; ch...  | 33            | 30.10.2021, ... |             |
| 1 | /css/chunk-vendors.5d3035bc.css?_WB_REVISION_=754...  | basic         | text/css; ch...  | 162,734       | 30.10.2021, ... |             |
| 2 | /index.html?_WB_REVISION_=bebc843a14749078a8049...    | basic         | text/html; c...  | 1,689         | 30.10.2021, ... |             |
| 3 | /js/app.865cd3a1.js?_WB_REVISION_=45d918f1fe82247...  | basic         | application/...  | 5,252         | 30.10.2021, ... |             |
| 4 | /js/chunk-vendors.a72f7e98.js?_WB_REVISION_=754547... | basic         | application/...  | 126,945       | 30.10.2021, ... |             |
| 5 | /manifest.json?_WB_REVISION_=f5f4e3b771427fc71f2b...  | basic         | application/...  | 559           | 30.10.2021, ... |             |
| 6 | /robots.txt?_WB_REVISION_=b6216d61c03e6ce0c9aea6...   | basic         | text/plain; c... | 24            | 30.10.2021, ... |             |

Cache Storage details:

```

{
  name: "PWA Demo Employees",
  short_name: "PWA Demo",
  theme_color: "#963484",
  background_color: "#28c2ff",
  display: "standalone",
  icons: [
    {src: ".../img/icons/android-chrome-192x192.png", sizes: "192x192", type: "image/png"},
    {src: ".../img/icons/android-chrome-512x512.png", sizes: "512x512", type: "image/png"},
    {src: ".../img/icons/android-chrome-maskable-192x192.png", sizes: "192x192", type: "image/png"},
    {src: ".../img/icons/android-chrome-maskable-512x512.png", sizes: "512x512", type: "image/png"}
  ],
  name: "PWA Demo Employees",
  short_name: "PWA Demo",
  start_url: ".",
  theme_color: "#963484"
}

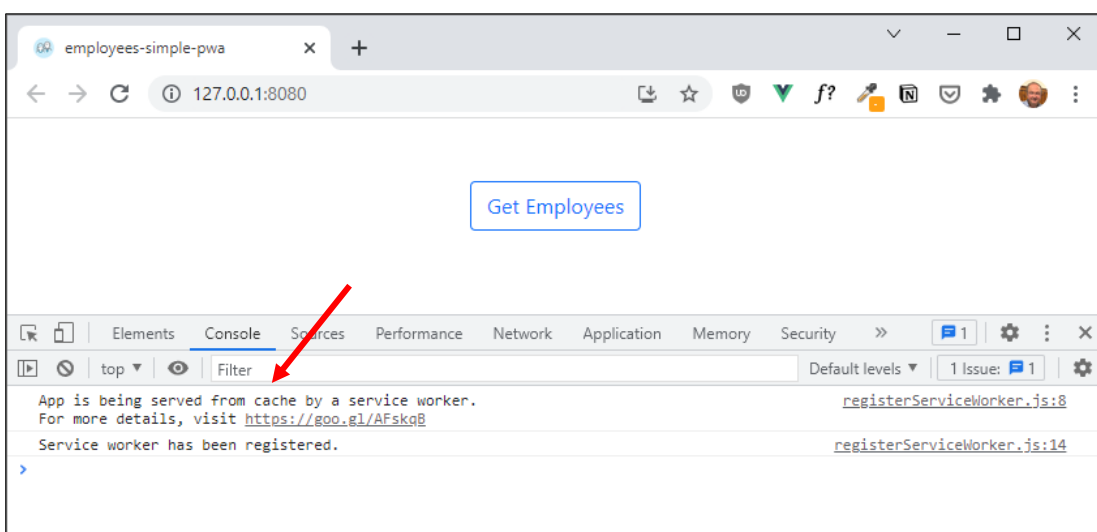
```

Total entries: 7

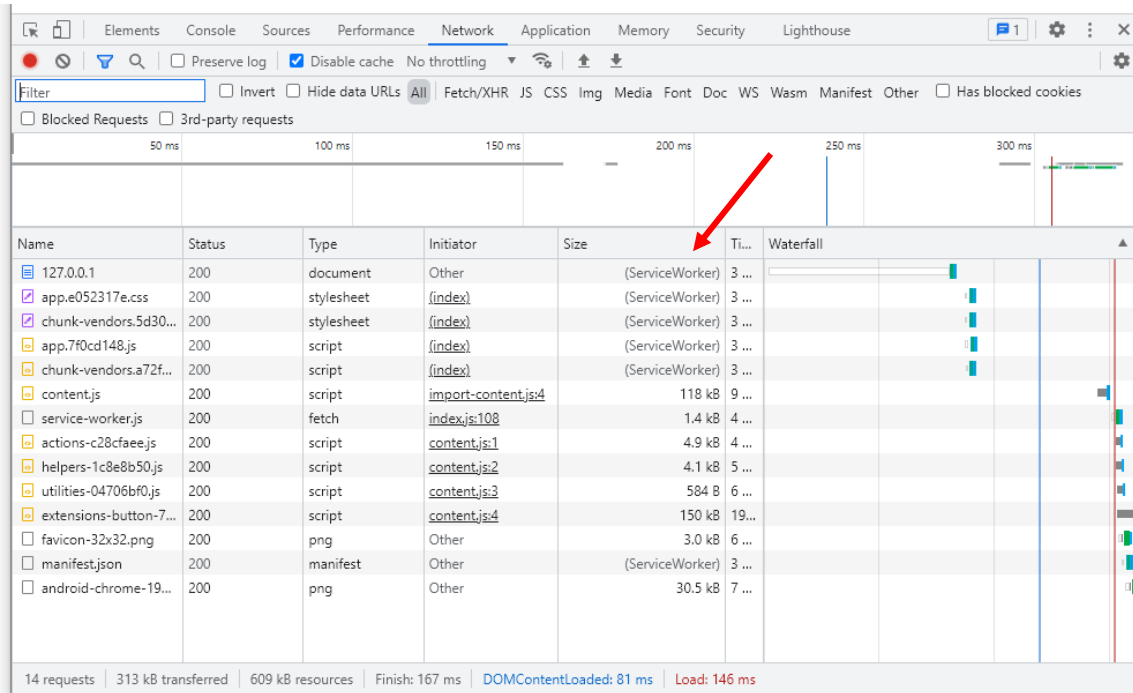
Die Service Worker Versionen werden durchnummeriert (hier läuft Nummer 6). Solange der Service Worker am Client läuft, holt er die Daten für den AppFrame aus dem Cache.

Drücke auf **Get Employees**. Das Holen der Daten ändert nichts am Cache! Der Default Service Worker kümmert sich nur um den App Frame.

Du siehst auch an einer Meldung in der Console und im Network Tab, dass der App Frame aus dem Cache kommt.



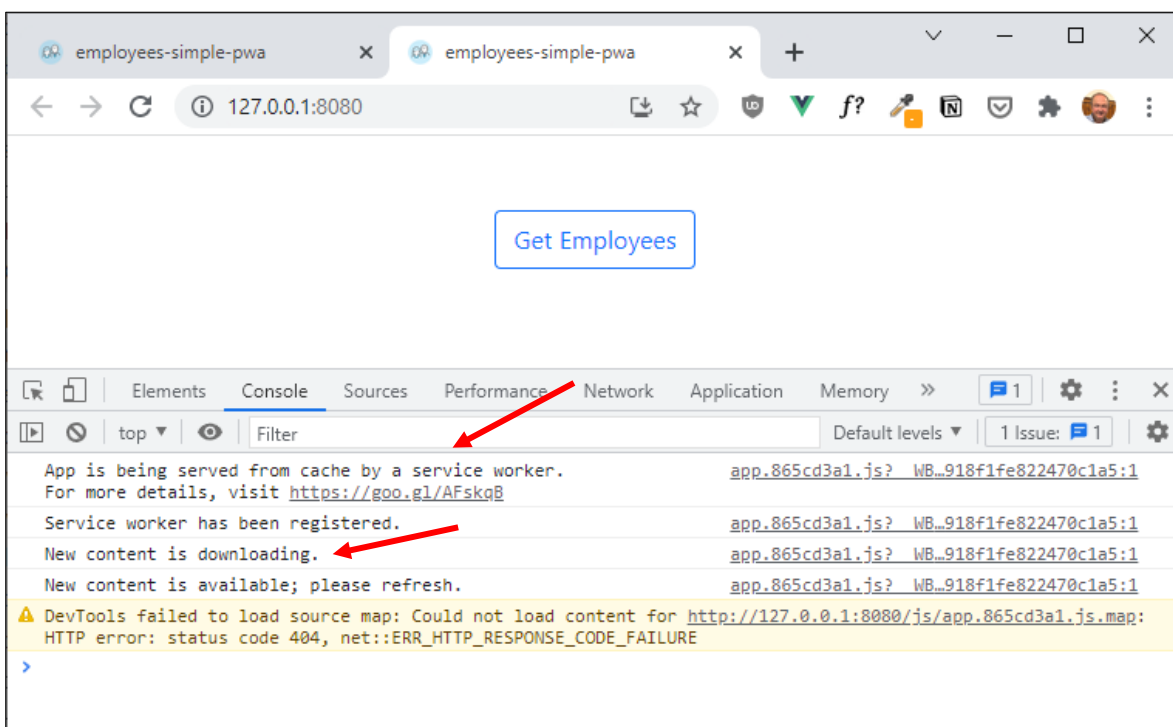
The screenshot shows the Chrome DevTools Console. A red arrow points to the message: "App is being served from cache by a service worker. For more details, visit <https://goo.gl/AFskqB>". Below this message, it says "Service worker has been registered." and provides a link to the `registerServiceWorker.js` file.



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 starte den **http-server** neu. Nichts ändert sich! Aber du bekommst einen Hinweis!



employees-simple-pwa

127.0.0.1:8080

Get Employees

App is being served from cache by a service worker.  
For more details, visit <https://goo.gl/AFskqB>

Service worker has been registered.

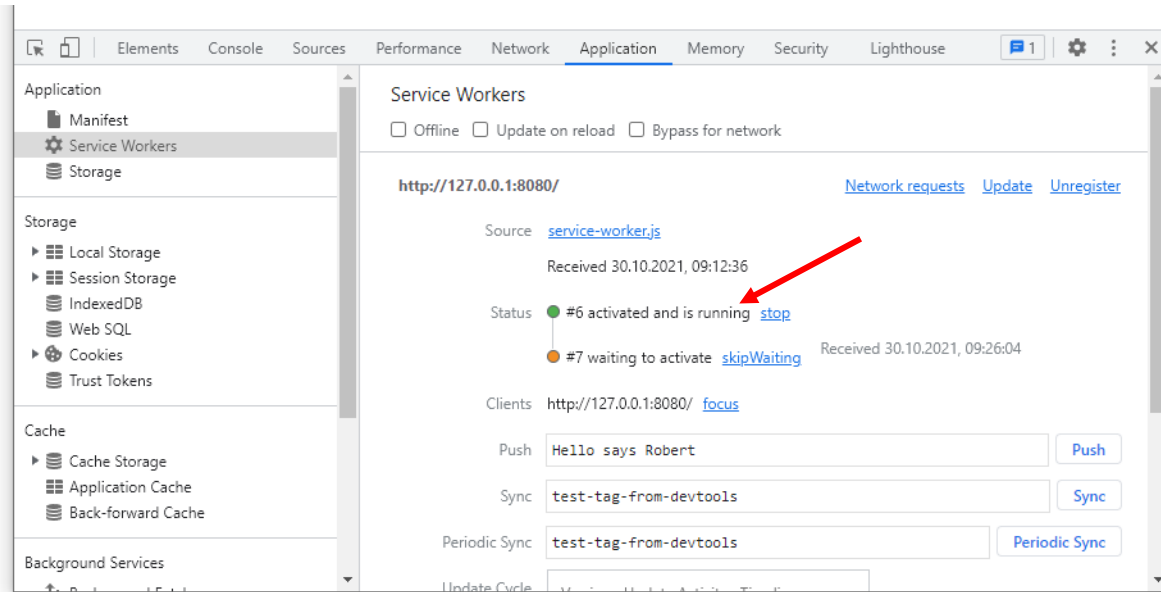
New content is downloading.

New content is available; please refresh.

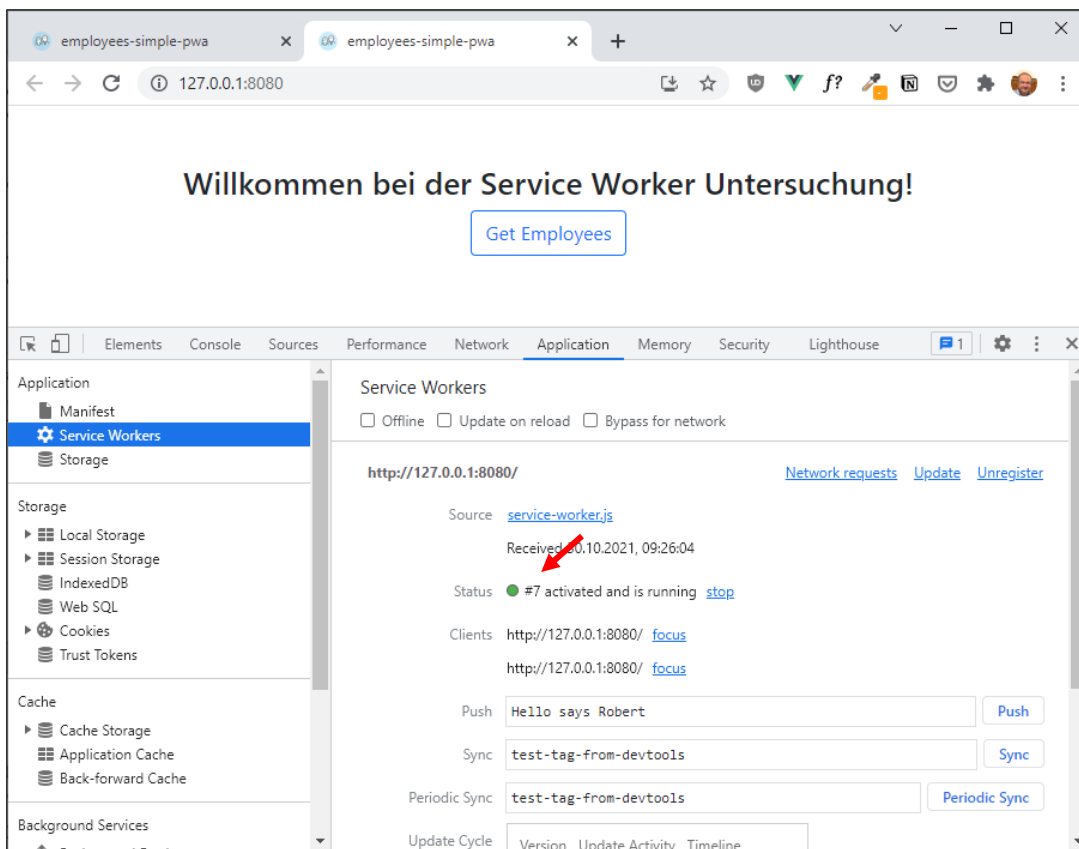
DevTools failed to load source map: Could not load content for <http://127.0.0.1:8080/js/app.865cd3a1.js.map>:  
HTTP error: status code 404, net::ERR\_HTTP\_RESPONSE\_CODE\_FAILURE

Robert Baumgartner

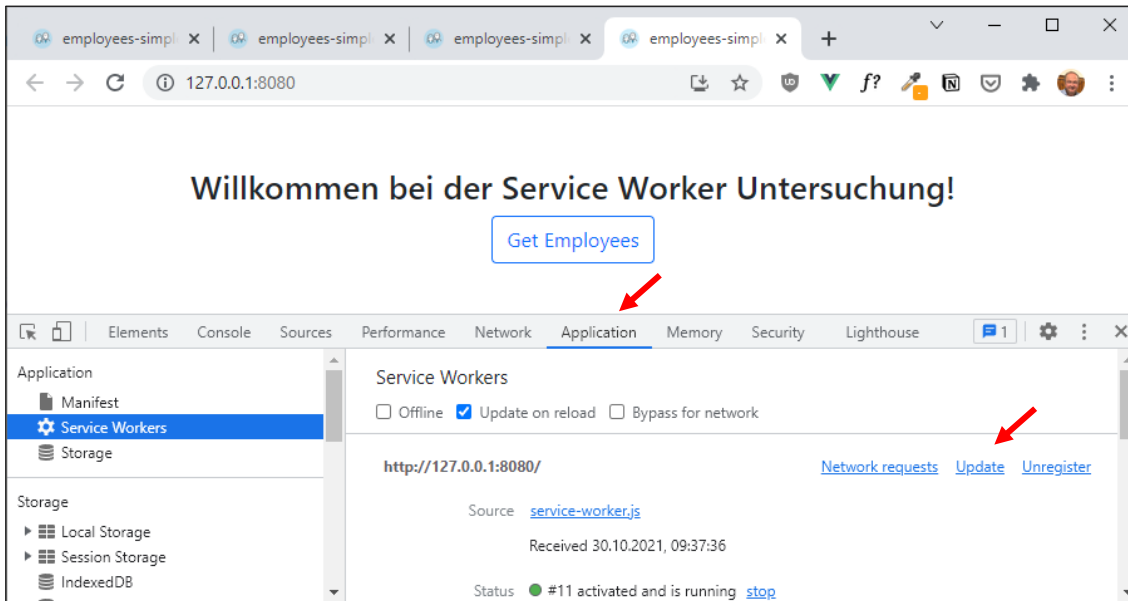
Auch ein simples Refresh nützt nichts, da durch den Build ein neuer Service Worker gebaut wurde (auch wenn er inhaltlich gleich ist), bleibt aus Sicherheitsgründen die alte Version des Service Workers noch aktiv, solange der Browser läuft.



Der Service Worker wird erst ersetzt, wenn der Browser geschlossen und neu gestartet wird bzw. wenn du in den Dev Tools den Service Worker ersetzt.



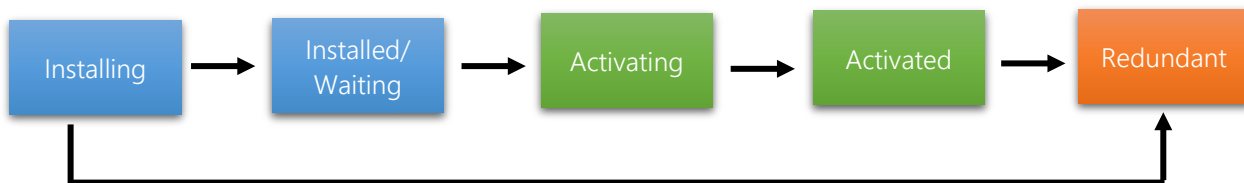
Sind die Dev Tools offen, kannst du **Update on reload** anklicken (erzeugt neue Versionen und ersetzt den alten Service worker) oder per Hand jedes Mal **Update** triggern.



OK. Es wird Zeit, sich den Lebenszyklus des Service Workers genauer anzusehen.

## Lebenszyklus des Service Workers

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

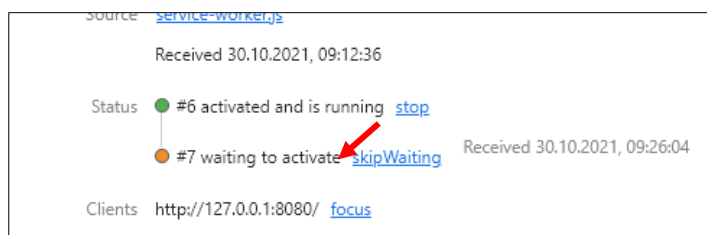


### Installing

Wenn ein neuer Service Worker registriert wird, dann wird vom Browser dessen JavaScript Code analysiert (geparst) und versucht ihn in den "Installed" Zustand zu versetzen. Wenn das erfolgreich war, ist der Service Worker "Installed". Tritt jedoch während des Vorgangs ein Fehler auf, geht der Service Worker in den "Redundant" Zustand über.

### Installed/Waiting

Ist der Service Worker im "Installed" Zustand, geht er sofort in den "Activating" Zustand über, es sei denn, ein anderer aktiver Service Worker ist gerade aktiv. In diesem Fall bleibt er im Zustand "Waiting".



### Activating

Der Service Worker ist im Begriff aktiviert zu werden.

## Activated

Der Service Worker ist aktiviert. Er kontrolliert nun die Seite.

## Redundant

Service Worker, die bei der Registrierung oder der Installation fehlgeschlagen sind oder durch neuere Versionen ersetzt werden, werden in den "Redundant" Status versetzt. Service Worker in diesem Zustand haben keine Auswirkungen mehr auf deine App und werden gelöscht.

## Das npm Modul register-service-worker

Vue.js verwendet **register-service-worker**, ein **npm** Modul, dass uns über wichtige Events im Lebenszyklus des Service Workers, informiert:

Wir können diese Events verwenden, um Nachrichten an das Client GUI zu senden.

```
ready () {  
  console.log(  
    'App is being served from cache by a service worker.\n' +  
    'For more details, visit https://goo.gl/AFskqB'  
  )  
},  
registered () {  
  console.log('Service worker has been registered.')  
},  
cached () {  
  console.log('Content has been cached for offline use.')  
},  
updatefound () {  
  console.log('New content is downloading.')  
},  
updated () {  
  console.log('New content is available; please refresh.')  
},  
offline () {  
  console.log('No internet connection found. App is running in offline mode.')  
},  
error (error) {  
  console.error('Error during service worker registration:', error)  
}
```

Unter anderem eine Information anzeigen, dass sich die Seite geändert hat.

**Aufgabe 4:** Führe die folgenden Schritte durch.

In **register-service-worker.js** ändere die **update** Function auf:



```
updated(registration) {  
  document.dispatchEvent(new CustomEvent('swUpdated', { detail: registration }));  
},
```

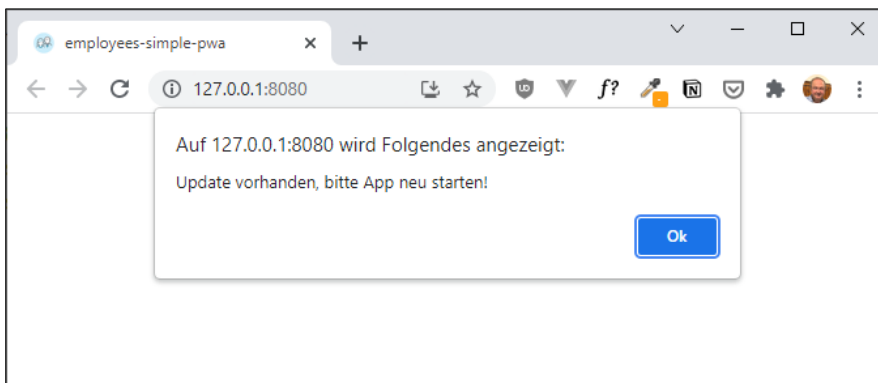
in **App.vue** registriere eine Funktion, die nur einmal aufgerufen werden soll, wenn der Event eintritt.

```
created() {  
  document.addEventListener('swUpdated', this.updateAvailable, { once: true });  
},
```

Zeige im Fall eines Updates (Methode **updateAvailable**) die Message im GUI an. Zum Beispiel mit einem Alert. Besser du verwendest aber die Bootstrap Klasse **alert** mit **alert-danger**. Sonst überlagert das Pop-up wie unten teilweise dein GUI.

Achtung: Dir sollte klar sein, dass der alte Service Worker alles aus dem Cache liefert. Wenn du also testen möchtest, dann musst du den neuen Service Worker laden und danach eine Änderung vornehmen. Dann wird sie dir angezeigt.

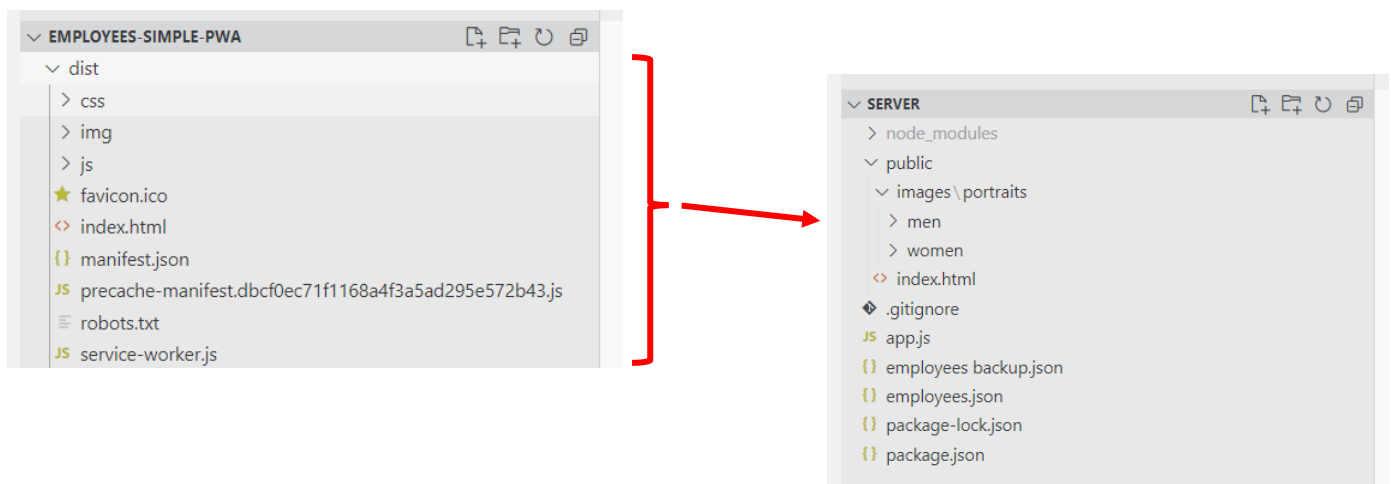
Die Update Meldung ist auch praktisch, um nicht selbst auf die gecachte Version hereinzufallen.



## Hosting Client und Server



In diesem Beispiel kombinieren wir den Server und den Build vom Client (im `\public` Verzeichnis des Servers) und hosten server plus Client auf **Heroku**.

Der einfachste Weg dazu ist, den Inhalt von `\dist` in das `\public` Verzeichnis des Servers zu kopieren.



Robert Baumgartner

Da dies von Hand aus auf Dauer mühsam ist, wollen wir das automatisieren. Sieh zu, dass Client und Server im Verzeichnis auf gleicher Ebene sind.

| Name   | Änderungsdatum   | Typ         |
|--|------------------|-------------|
|  employees-simple-pwa | 30.10.2021 16:11 | Dateiordner |
|  server               | 29.10.2021 17:36 | Dateiordner |

**Aufgabe 5:** Führe die folgenden Schritte durch.

Im Client ergänze den **vue.config.js** File.

```
const path = require('path');

module.exports = {
  outputDir: path.resolve(__dirname, '../server/public'),
  // ...
};
```

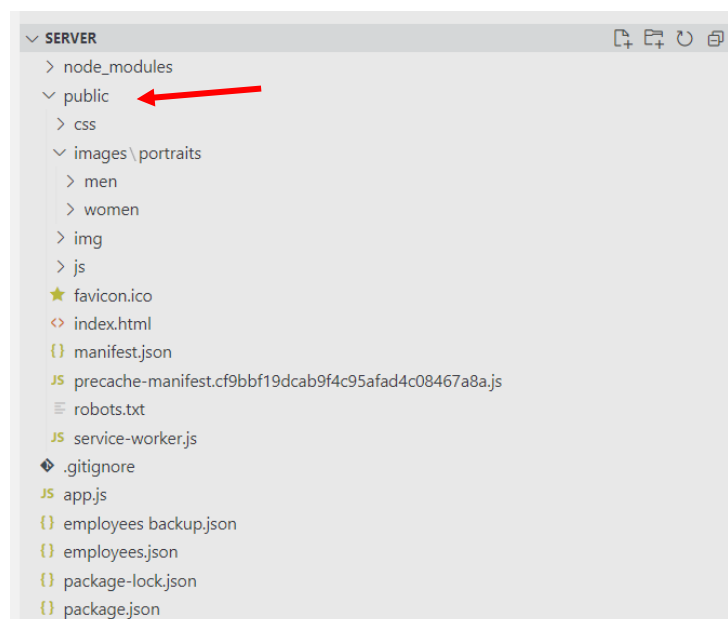
Damit uns die Images nicht verloren gehen, ändere **package.json** im Client und füge **--no-clean** zum Build Script hinzu.

```
"scripts": {
  "serve": "vue-cli-service serve",
  "build": "vue-cli-service build --no-clean",
  "start": "http-server ./dist",
  "lint": "vue-cli-service lint"
},
```

Ändere nun den **.env.production** File. Da wir den Client vom Server laden, benötigen wir keine Server Adresse und verwenden einfach die relativen URLs der Daten.

**VUE\_APP\_SERVER=**

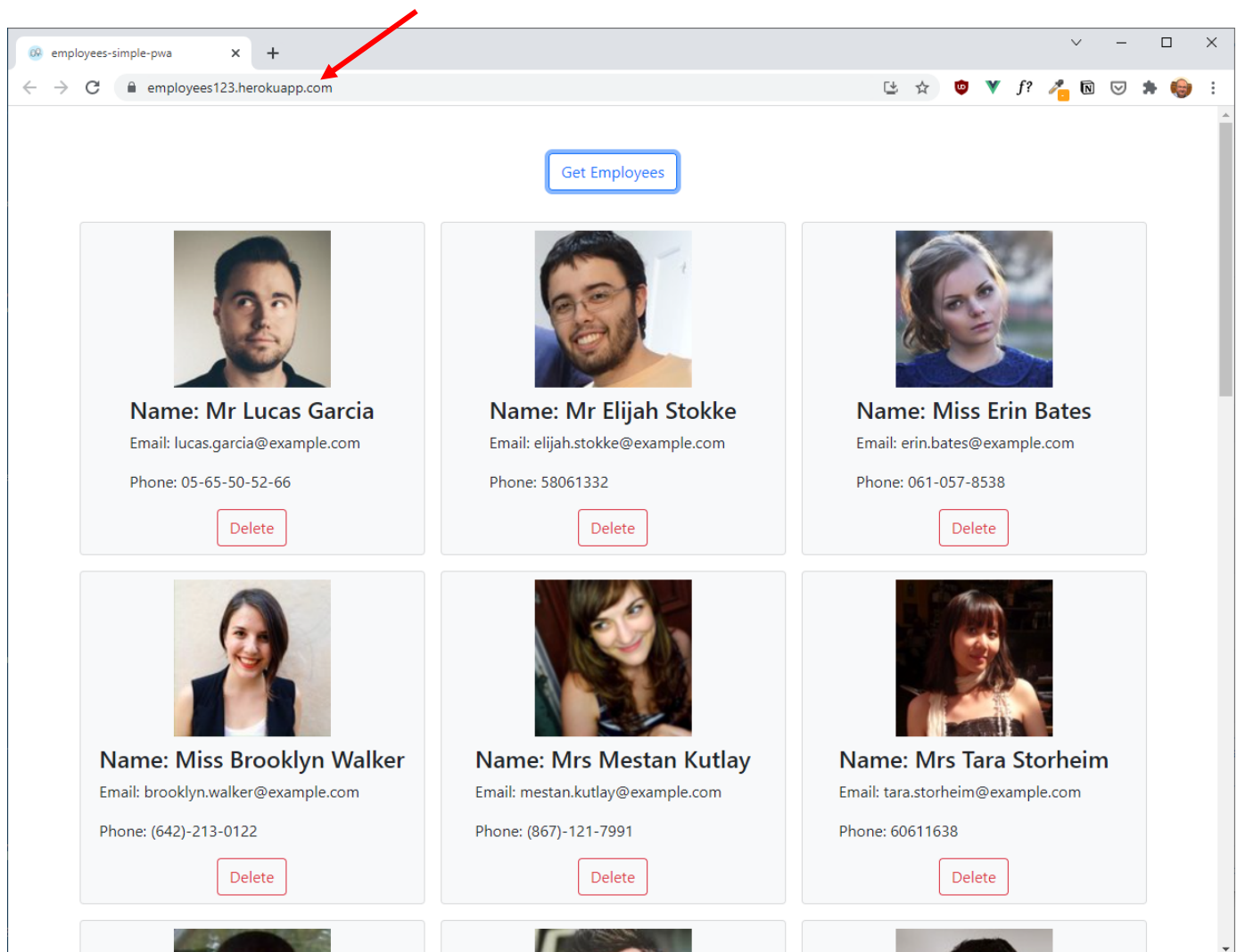
Nach dem Build ist das Ergebnis:



Robert Baumgartner

**Aufgabe 6:** Hoste den Server (mit dem Clientbuild in `/public`) auf **Heroku**. Wenn du nicht weißt, wie das geht, lese das Arbeitsblatt **Hosten deiner App auf Heroku.pdf**

```
remote: -----> Build succeeded!
remote: -----> Discovering process types
remote:      Procfile declares types      -> (none)
remote:      Default types for buildpack -> web
remote:
remote: -----> Compressing...
remote:      Done: 35.4M
remote: -----> Launching...
remote:      Released v3
remote:      https://employees123.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/employees123.git
* [new branch]      master -> master
```



**Aufgabe 7:** Besuche deine Seite mit deinem Handy an und füge die PWA als App hinzu!

Beispiel nächste Seite!

