

Vorbereitung

Aufgabe 1: Entpacke die Datei **server_START.zip** und ersetze den alten Server durch den neuen Server. Dieser hat zwei Verzeichnisse für statische Routen.

```
app.use(express.static(path.join(__dirname, '/public')));  
app.use(express.static(path.join(__dirname, '/client')));
```

In **/client** wird der Build des Clients gespeichert und in **/public** befinden sich die Bilder. Ändere beim Vue Client **vue.config.js** entsprechend und entferne aus **package.json** **--no-clean**. Die Bilder können nun nicht mehr überschrieben werden! Teste den Server.

Der ServiceWorkerGlobalScope

Wie du weißt, gibt es im Browser das globale **window** Objekt. Es stellt ein Fenster dar, das eine Webseite enthält, die wiederum durch das **document** Objekt repräsentiert wird. Ein Fenster kann dabei ein neues Fenster, ein neuer Tab, ein Frameset oder ein einzelner Frame sein, der mit JavaScript erstellt wurde.¹ Jedes Mal, wenn du ein neues Fenster oder einen neuen Tab öffnest, wird automatisch ein **window** Objekt erstellt, das dieses Fenster bzw. diesen Tab repräsentiert.

Innerhalb eines Service Workers macht aber das Konzept eines Fensters keinen Sinn, und daher ist sein globales Objekt eine Instanz von **ServiceWorkerGlobalScope**. Anstelle von **window** ist dieses Objekt über das Schlüsselwort **self** zugänglich. Mit **self** kannst du zum Beispiel auf Events wie **Push** lauschen, oder dem Service Worker sagen, dass er sich ungeachtet eines bereits laufenden Service Workers aktivieren soll (**skipWaiting**).

Lies: <https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerGlobalScope>

Aufgabe 2: Baue in **service-workers.js** ein:

```
self.skipWaiting();
```

und erweitere im Vue Client die **updateAvailable** Methode, sodass die Seite vom User gleich neu geladen werden kann:

```
updateAvailable() {  
  if (confirm(`New content is available!. Click OK to refresh`)) {  
    window.location.reload();  
  }  
},
```

Test, siehe nächste Seite!

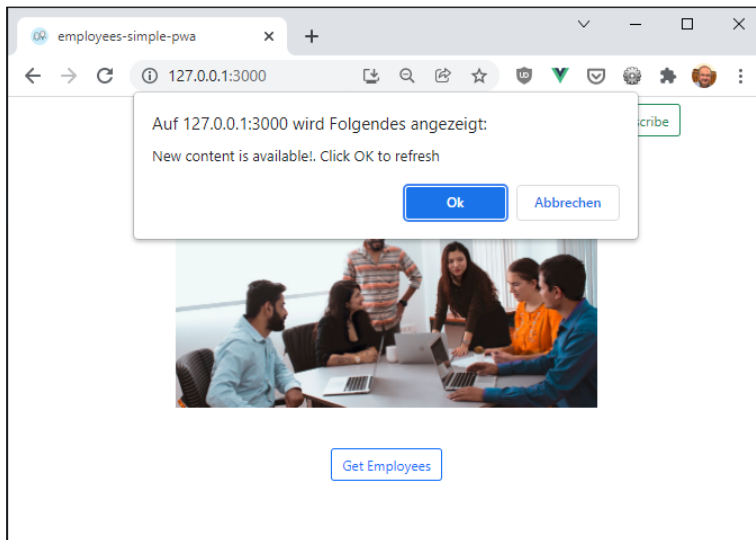
Offline/Online Anzeige

Im letzten Arbeitsblatt haben wir uns um das Caching gekümmert. Daten und Bilder stehen nun auch offline zur Verfügung. Aber ein Problem gibt es noch mit dem **Delete** Button, denn wenn die App offline ist, ergibt es keinen Sinn (ohne entsprechendes lokales Caching), einen Delete Request abzusetzen!

Wir wollen daher dem User (wie bei Spotify oder YouTube) anzeigen, dass er offline ist. Zusätzlich soll in diesem Fall der **Delete** Button nicht mehr angezeigt (bzw. disabled) werden.

¹ Bei einem Browser mit mehreren Tabs repräsentiert ein **window** Objekt einen einzelnen Tab, aber einige seiner Eigenschaften wie **innerHeight**, **innerWidth** und Methoden wie **resizeTo()** betreffen das gesamte Browserfenster.

Robert Baumgartner



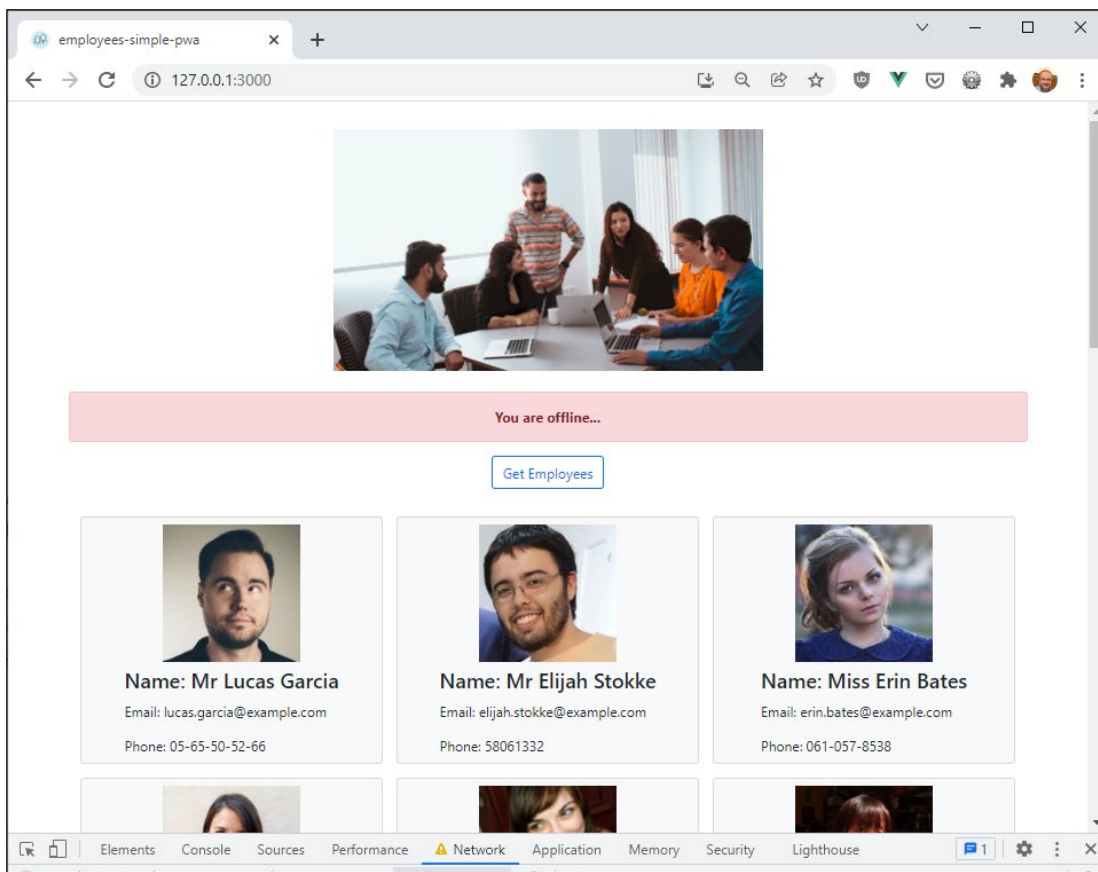
Um den Offline Status anzuzeigen, verwenden wir ein Bootstrap Alert. Ein Alert ist eine einfache Möglichkeit, vordefinierte Warnmeldungen zu erstellen. Siehe: <https://getbootstrap.com/docs/5.0/components/alerts/>.

Um den Status feststellen zu können, gibt es die Events **online** und **offline**.

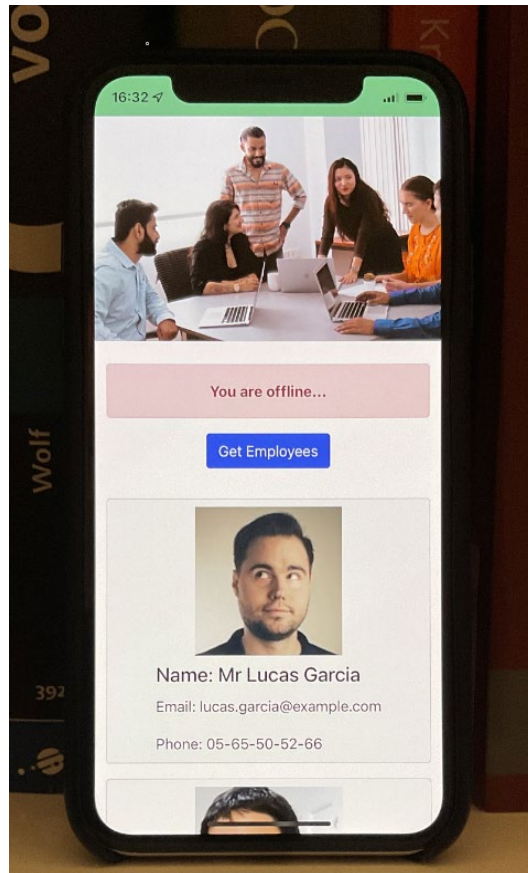
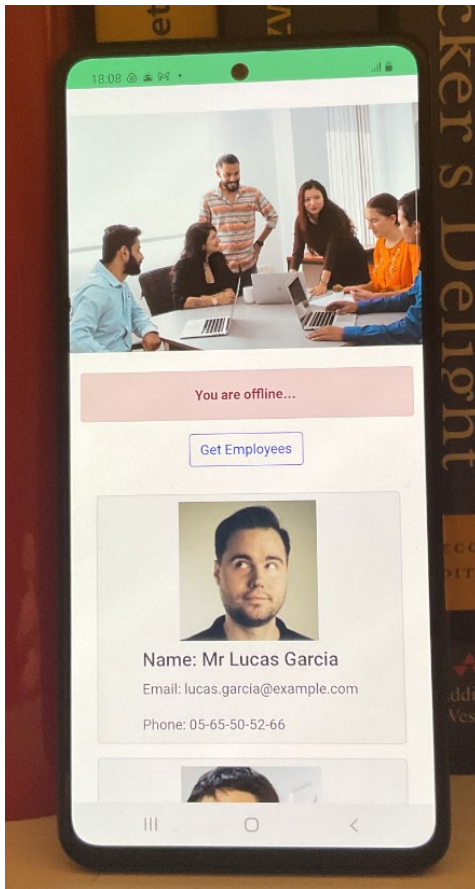
Aufgabe 3: Verwende die untenstehenden Events im **created** Hook und eine Variable **offline**:

```
window.addEventListener('online', () => (this.offline = false));  
window.addEventListener('offline', () => (this.offline = true));
```

Zeige den Offline Status im GUI an.



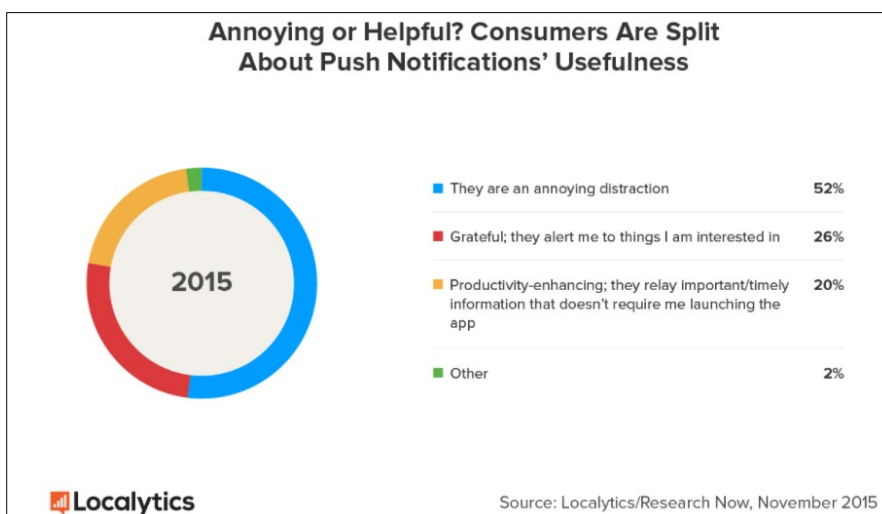
Robert Baumgartner

Aufgabe 4: Lade deine App auf Heroku hoch und teste mit dem Handy.


Push Notifications

Push-Notifications sind kurze Mitteilungen, bestehend aus Text und Icons, die unabhängig von der Nutzung einer Webseite oder App von einem Server an einen Benutzer geschickt werden. Allerdings muss der User vorher zugestimmt haben, ob er der Zusendung von Mitteilungen zustimmt. Diese Zustimmung kann jederzeit widerrufen werden. Daher sollen Push-Notifications für den User einen Mehrwert darstellen, indem sie zusätzliche Informationen liefern oder an bekannte (wichtige) Informationen erinnern.

Allerdings: Über 50 % der User empfinden Push-Benachrichtigungen als störend:

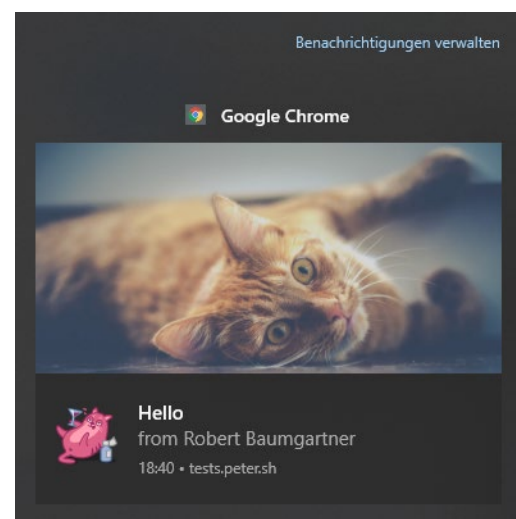
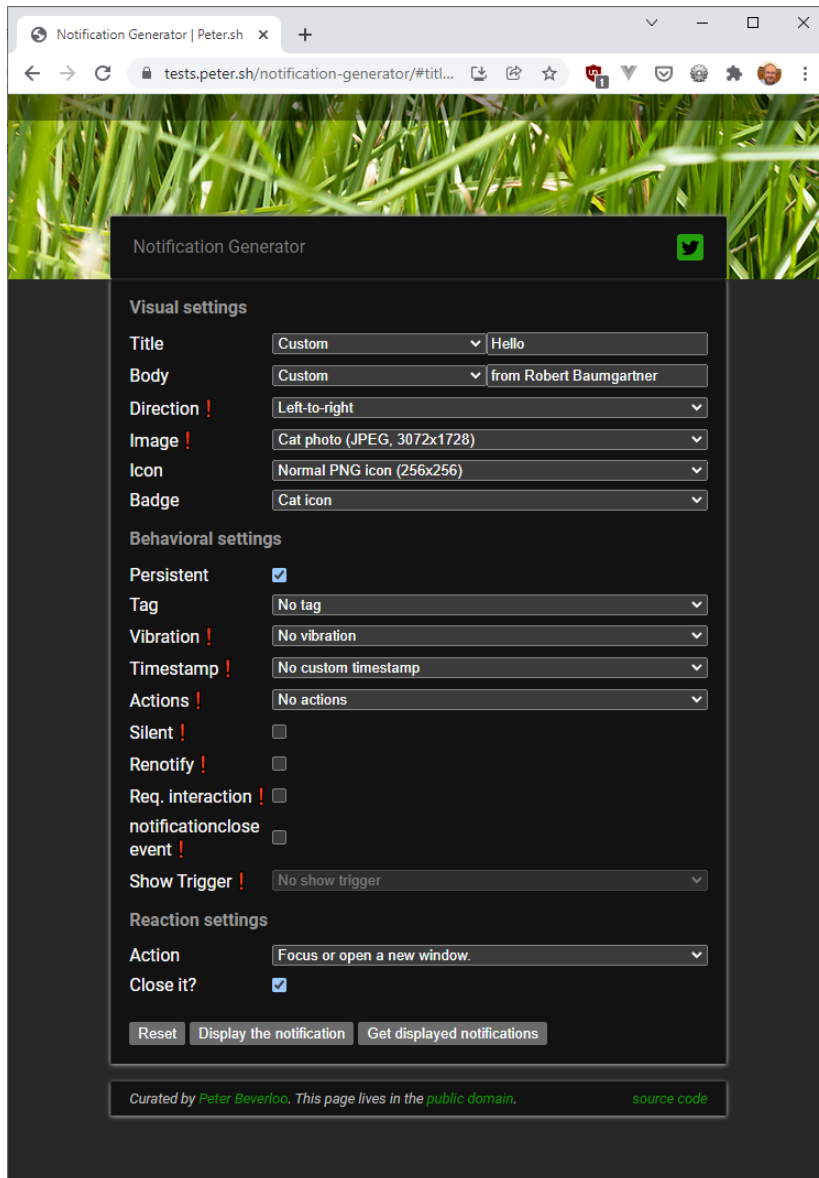


Robert Baumgartner

Es kommt eben immer darauf an, wie relevant die Nachricht für den User ist. Die Grenze zum E-Mail Spam ist schmal.

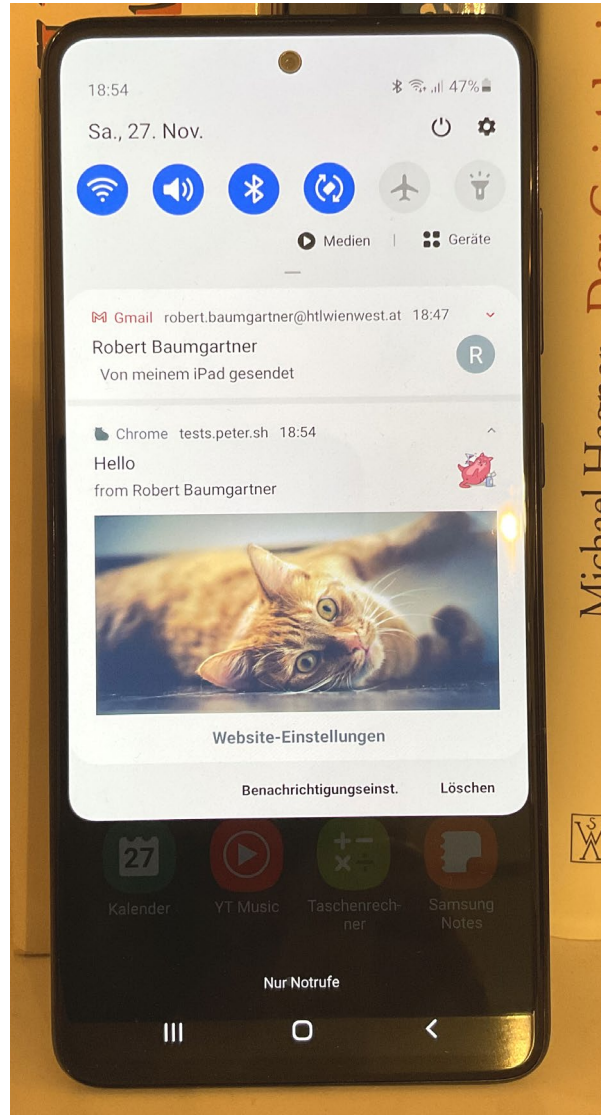
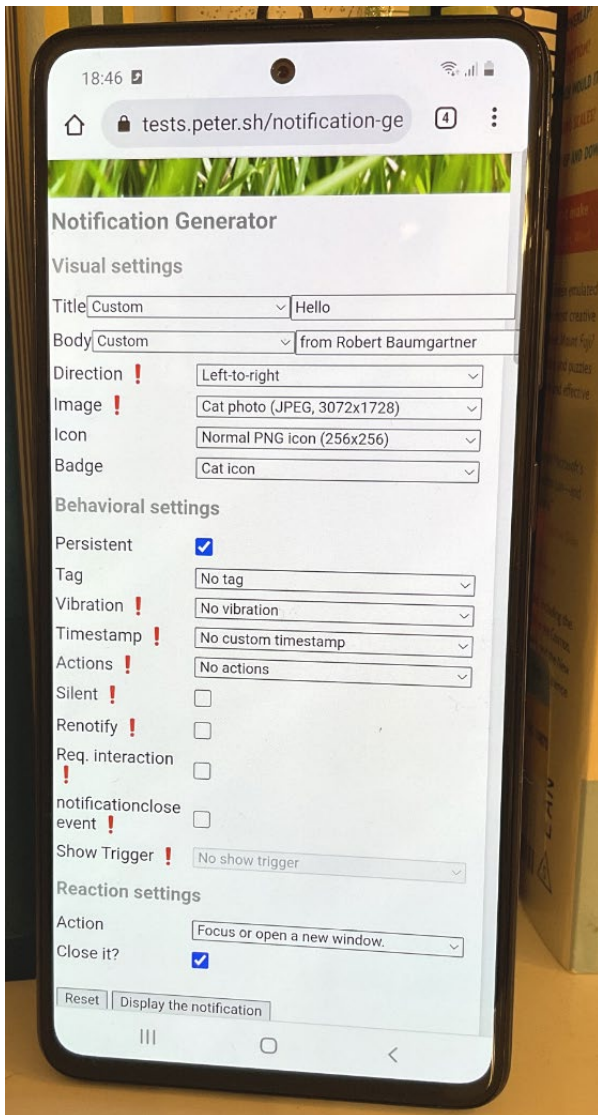
Leider sind Web Push Nachrichten auf iOS (15.1) nicht möglich. Apple hält sich nicht an den Standard und hat seine eigenen Push Nachrichten. Siehe: https://en.wikipedia.org/wiki/Apple_Push_Notification_service

Aufgabe 5 (geht nicht unter iOS): Zur Einstimmung benutze den Notification Generator von Peter Beverloo: <https://tests.peter.sh/notification-generator/>

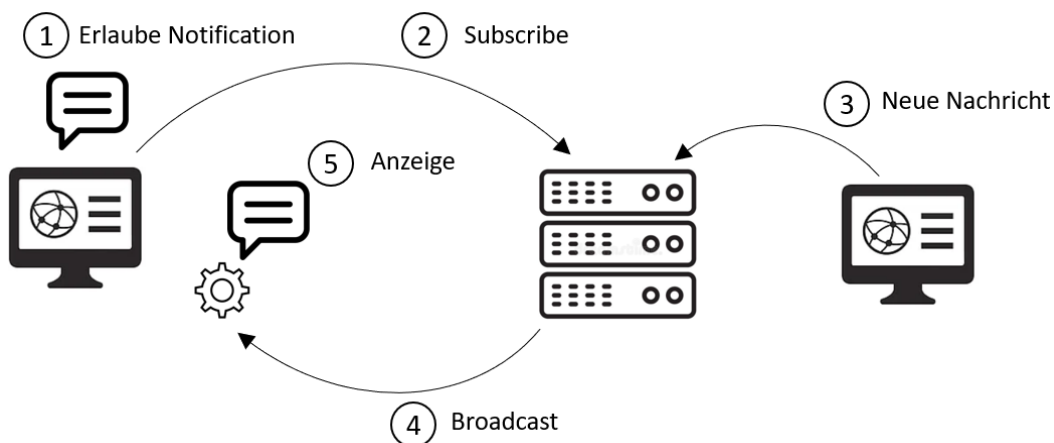


Beachte: Windows zeigt die Push Nachrichten im Aktionscenter (Taskleiste rechts unten an).

Android: siehe nächste Seite!



Wie ist der Ablauf bei Web Push Notifications? Es gibt mehrere Komponenten, die hier zusammenspielen, wie im folgenden Diagramm deutlich wird:



1. Wenn der Client zum ersten Mal von einem Server geladen wird, kann der User gefragt werden, ob er der Zusendung von Mitteilungen zustimmt (Firefox und Safari verlangen eine User Aktion, daher verwenden wir in unserem Beispiel einen **Subscribe** Button). Lehnt der User ab, bekommt er keine Push Nachrichten. Dies kann später widerrufen werden.

Robert Baumgartner

2. Nach dem Registrieren des Service Workers meldet sich der Client beim Push Service an. Dazu wird in unserem Fall ein **subscription** Objekt an die POST Route **/subscribe** am Server gesendet. Es enthält neben optionalen Einstellungen den public Key des Push Services, das auf dem Server läuft. Das Subscription Objekt wird am Server gespeichert (normalerweise wird dazu eine Datenbank verwendet, wir verwenden ein Array).
3. Der Server bekommt irgendwann von einem Admin Client eine Nachricht zum Verteilen. In unserem Fall wird dazu eine Nachricht an die POST Route **/notify** des Push Services gesendet. Beachte, dass die Route und der Inhalt der Einfachheit halber nicht abgesichert sind!
4. Der Server schickt die Nachricht an alle, die sich angemeldet haben. Er geht dazu in einer Schleife alle gespeicherten **subscription** Objekte durch.
5. Der Service Worker empfängt einen push Event und zeigt ihn über das Notification Service des Gerätes an.

Aufgabe 6: Führe die folgenden Schritte durch (geht nicht unter iOS, unter macOS geht's mit Chrome)

Zunächst erweitere den Server. Wir verwenden das npm modul **web-push**.

<https://github.com/web-push-libs/web-push>

Installiere **web-push** und importiere die Library in die Datei **index.js** im Verzeichnis **/routes**.

Danach generiere die VAPID (**V**oluntary **A**pplication Server **I**dentification) Keys. Das ist ein public/private Key Paar. Die Keys sind seitens des Push Protokolls nicht vorgeschrieben (deswegen voluntary 😊), aber **web-push** verlangt sie.

```
$ "node_modules\\.bin\\web-push" generate-vapid-keys
```

```
=====
```

Public Key:

B0yA6dXH071UjG4B48o4pJ9GkW4XypWh_9YamMCnJ3ug_tofRG0z900rKsRjXIjd1Aa5TFCzGgvzSy1Ly

Private Key:

HD4LJyX7X6XbINCqDwDRAYQdeHfPjD00tRFZts09HQU

```
=====
```

Kopiere deine generierten Keys und gib sie in den **.env** file. Zum Beispiel:

PORT=3000

PUBLIC_VAPID_KEY=B0yA6dXH071UjG4B48o4pJ9GkW4XypWh_9YamMCnJ3ug_tofRG0z900rKsRjXIjd1. . .

PRIVATE_VAPID_KEY=HD4LJyX7X6XbINCqDwDRAYQdeHfPjD00tRFZts09HQU

Erstelle nun in der Datei **index.js** im Verzeichnis **/routes**:

a) Das Setup für **web-push** (siehe auch **web-push** Doku, Link oben)

```
const publicVapidKey = process.env.PUBLIC_VAPID_KEY;
const privateVapidKey = process.env.PRIVATE_VAPID_KEY;

webpush.setVapidDetails(
  'mailto:robert.baumgartner@htlwienwest.at',
  publicVapidKey,
  privateVapidKey,
);
```

Robert Baumgartner

b) Die POST Route für **/subscribe** (das Array heißt **subscriptions**):

```
subscription.push(req.body);  
res.status(201).end();
```

c) Die POST Route für **/notify**. Das Property **title** ist frei wählbar. Hier: **Push Test**. Wir schicken das, was wir im **req.body** bekommen, im **body** Property der Push Nachricht weiter.

```
router.post('/notify', (req) => {  
  const payload = JSON.stringify({ title: 'Push Test', body: req.body });  
  for (const sub of subscription) {  
    try {  
      webpush.sendNotification(sub, payload);  
    } catch (error) {  
      console.error(error);  
    }  
  }  
});
```

Fertig ist der (ungesicherte) Server.

Nun zum Client:

In **App.js** erfolgt die **subscribe** Meldung an den Server, nachdem der User auf den Button **Subscribe** gedrückt hat.

```
async subscribe() {  
  if (!('serviceWorker' in navigator)) {  
    console.log('no service worker!');  
    return;  
  }  
  const registration = await navigator.serviceWorker.ready;  
  
  const subscription = await registration.pushManager.subscribe({  
    userVisibleOnly: true,  
    applicationServerKey: this.urlBase64ToUint8Array(publicVapidKey),  
  });  
  
  await axios.post('/subscribe', subscription);  
},
```

Das **subscription** Objekt enthält den public Key und die Option **userVisibleOnly: true**. Für den Browser muss das Property **userVisibleOnly** auf **true** gesetzt sein. Das bedeutet, dass du nicht im Hintergrund ohne das Wissen des Users Push Nachrichten senden darfst. Etwas, das als **Silent Push** bezeichnet wird.

Chrome und Edge werfen eine Exception (nächste Seite!), Firefox ignoriert das Setting. Es ist immer (implizit) **true**.

```

workbox Precaching is responding to: /service-worker.js logger.mjs:44
workbox Precaching is responding to: /manifest.json logger.mjs:44
❌ DOMException: Registration failed - permission denied vue.runtime.esm.js:1897
❌ Chrome currently only supports the Push API for subscriptions that will result in user-visible
messages. You can indicate this by calling pushManager.subscribe({userVisibleOnly: true}) instead. See https://go
o.gl/yqv404 for more details. 127.0.0.1/:1
>

```

Silent Push ist unter Android und iOS erlaubt. Es kann aber auch verwendet werden, um den User auszuspionieren. So kann Silent Push eine App aus dem Zustand "Angehalten" oder "Nicht ausgeführt" aufwecken, um Inhalte zu aktualisieren oder bestimmte Aufgaben auszuführen, ohne den User zu benachrichtigen (Geofencing, Tracking). Es kann auch leicht ermittelt werden, ob deine App deinstalliert wurde. Wie das geht? Nun: Wenn du keine Silent Push Nachricht an ein Gerät senden kannst, nachdem du vor ein paar Stunden eine Push-Benachrichtigung gesendet hast, kannst du ziemlich sicher sein, dass der Nutzer deine App deinstalliert hat.

Es gibt noch eine weitere Sache, die wir tun müssen. Der öffentliche Schlüssel, der ein Base64-String ist, muss in ein Uint8Array umgewandelt werden. In der Dokumentation (siehe Link oben) des npm-Pakets **web-push** wird gezeigt, wie man das macht. Ich habe den Code kopiert 😊.

```

urlBase64ToUint8Array(base64String) {
  const padding = '='.repeat((4 - (base64String.length % 4)) % 4);
  const base64 = (base64String + padding).replace(/-/g, '+').replace(/_/g, '/');

  const rawData = window.atob(base64);
  const outputArray = new Uint8Array(rawData.length);

  for (let i = 0; i < rawData.length; ++i) {
    outputArray[i] = rawData.charCodeAt(i);
  }
  return outputArray;
},

```

Endlich kann das Objekt zu unserer Route am Server geschickt werden.

Nun müssen wir noch im Service Worker das **push** Event Handling einbauen. Zum Beispiel:

```

self.addEventListener('push', event => {
  const data = event.data.json();
  self.registration.showNotification(data.title, {
    body: data.body.message,
    icon: 'img/icons/employees_192x192.png',
  });
});

```

Für die Notification gibt es noch mehr Optionen. Siehe:

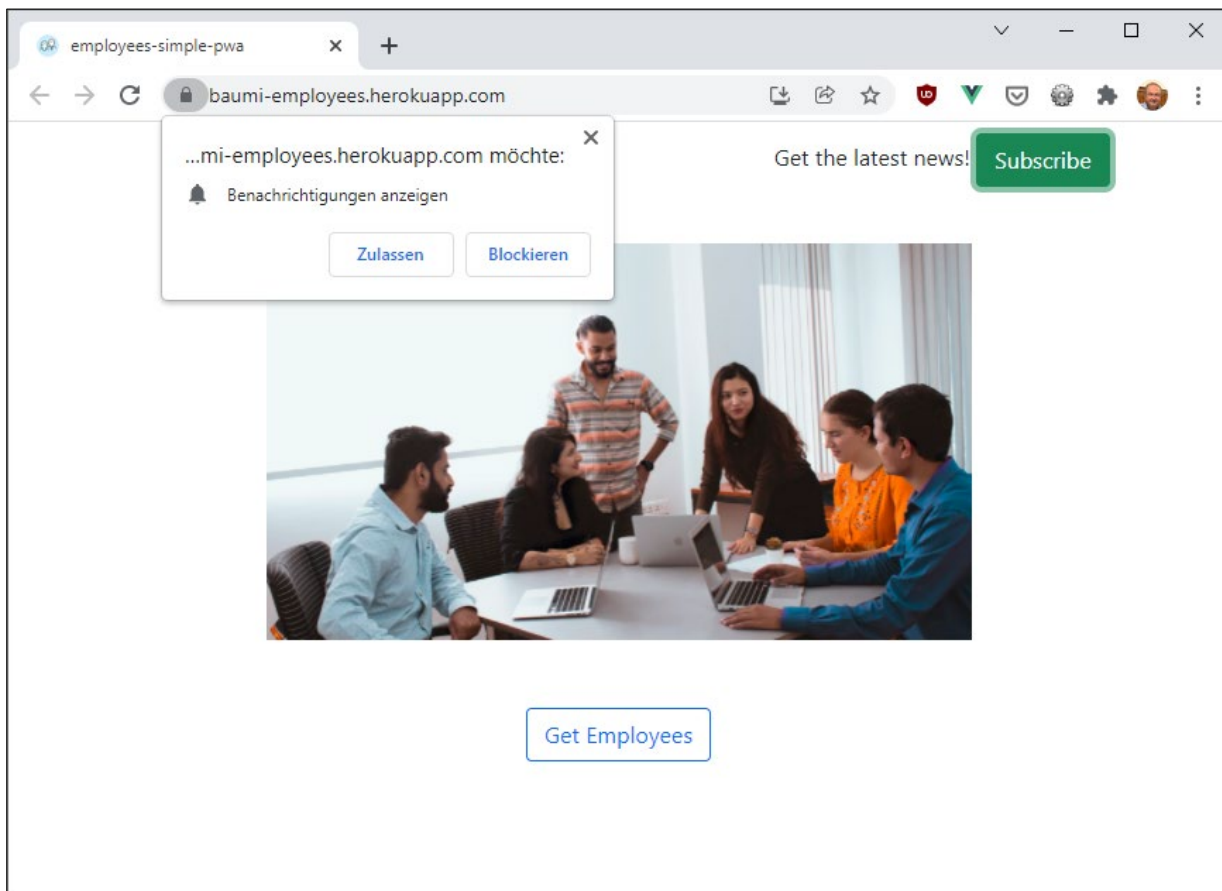
<https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerRegistration/showNotification>

OK nun ist alles fertig.

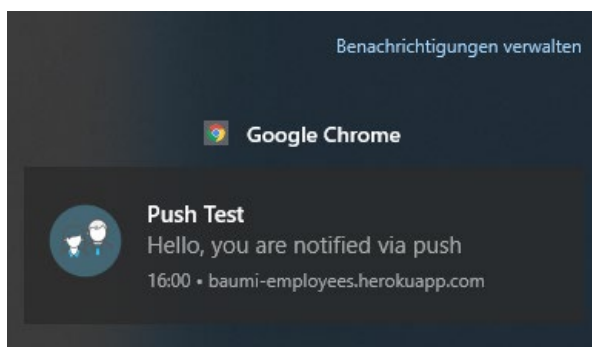
Die Screenshots der Tests findest du auf den nächsten Seiten.

Aufgabe 7: Hoste deine App auf Heroku und mache ebenfalls Screenshots von deinen Tests!

Robert Baumgartner

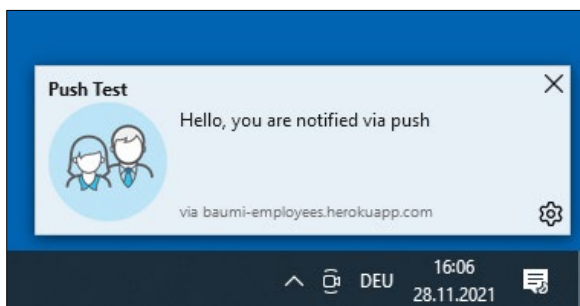
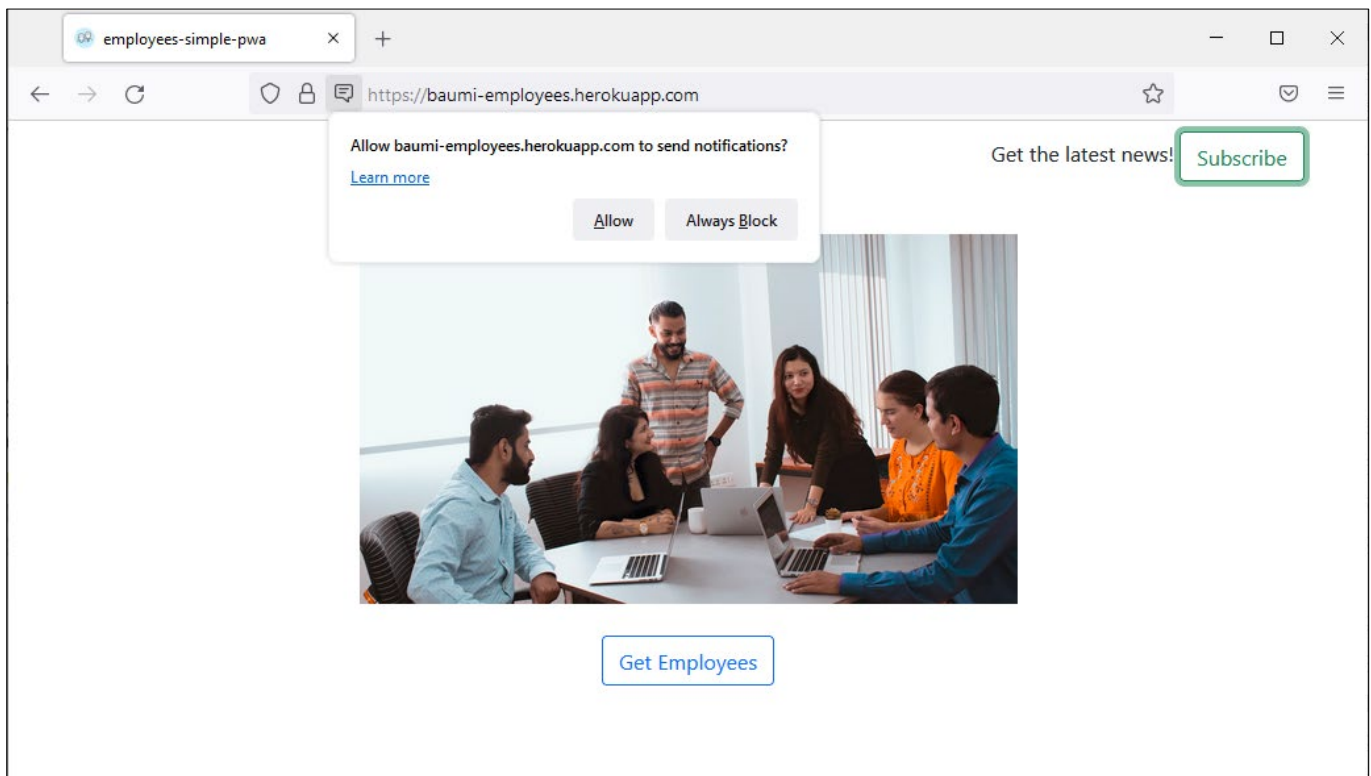
Der Test mit **Chrome**:

<pre>Send Request 1 POST http://baumi-employees.herokuapp.com/notify 2 Content-Type: application/json 3 4 { 5 "message": "Hello, you are notified via push" 6 }</pre>	<pre>1 HTTP/1.1 200 OK 2 Server: Cowboy 3 Connection: close 4 Content-Security-Policy: default-src 'self'; base-uri 'self';block-all-mixed-content;font- -src 'self' https: data:;frame-ancestors 'se lf';img-src 'self' data:;object-src 'none';s cript-src 'self';script-src-attr 'none';styl</pre>
---	---



Robert Baumgartner

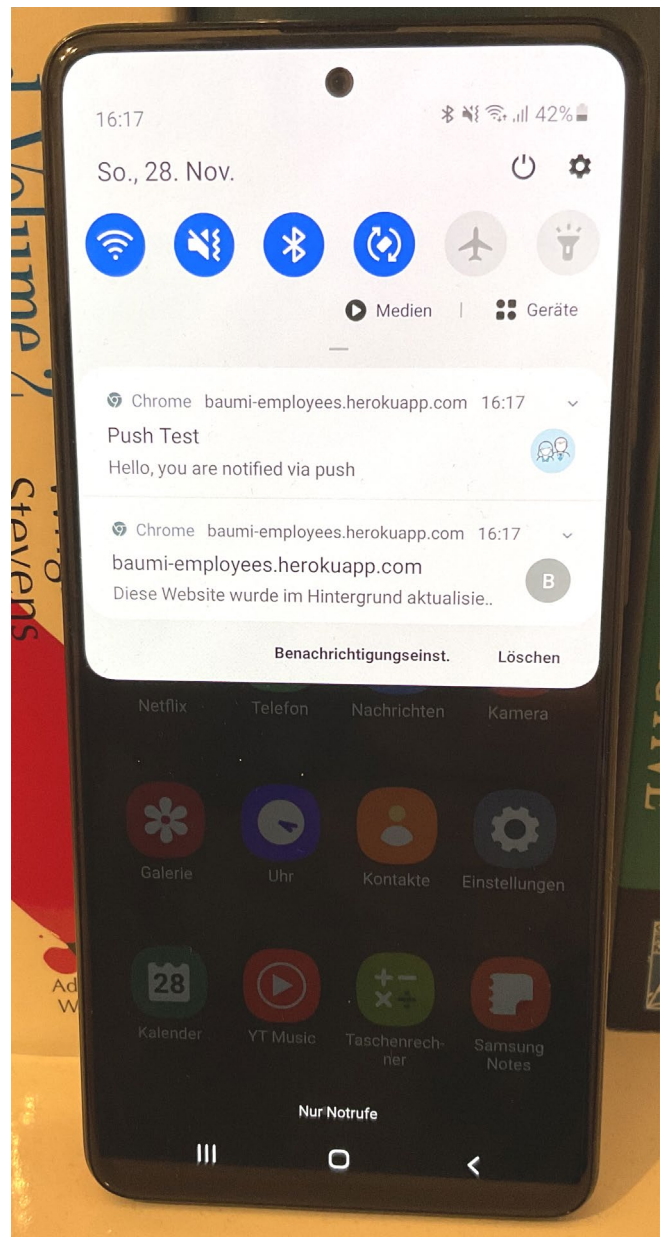
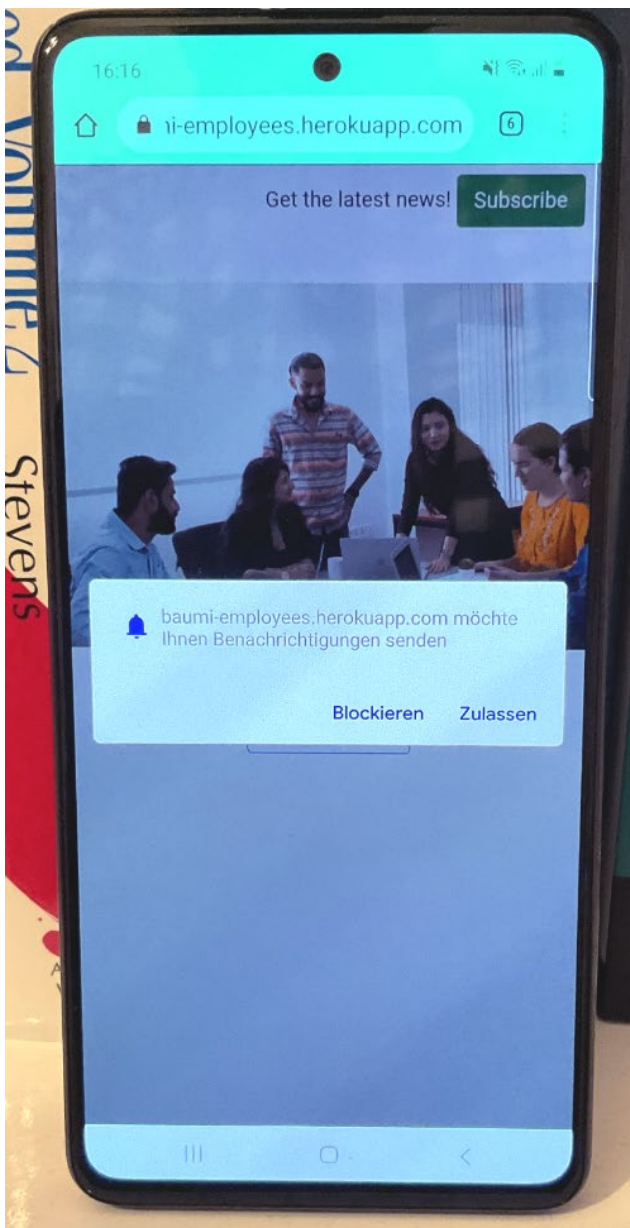
Der Test mit **Firefox**:



Jetzt müsste ich nur noch wissen, wieso die Notification von Firefox anders aussieht als bei Chrome! 😊

Robert Baumgartner

Der Test mit **Android**:



Der Test mit **MacOS Monterey**:

