

# PWA

## Bouw zelf een Progressive Web App met JavaScript

YOU NEED A NATIVE APP! Dat is wat ons herhaaldelijk is verteld sinds Apple voor het eerst de iPhone App Store aankondigde. En misschien is dat zo! Native apps kunnen een goede keuze zijn, afhankelijk van de grootte en behoeften van een organisatie.

Maar hoe zit het met potentiële klanten die je app niet hebben? Of huidige klanten op een desktop computer? Hoe zit het met mensen met beperkte ruimte op hun telefoons die apps verwijderen om ruimte te maken voor andere dingen? Dit is waar progressive web-apps (ook wel PWA's genoemd) van pas komen. Ze combineren de beste functies van het web met functies die voorheen alleen beschikbaar waren voor native apps. PWA's kunnen worden gestart via een icon. De apps laden onmiddellijk en kunnen worden gebouwd om offline te werken. De eerste keer dat iemand je website bezoekt, zijn de functies van een PWA onmiddellijk beschikbaar. Je hoeft de app niet te downloaden. Het is niet meer nodig een app uit een app-store te downloaden.

### Project Setup

Om te starten dien je eerst *node* te installeren: <https://nodejs.org/en/download/>. Als uitgangspunt voor de tutorial, clone je deze Github repository:

```
git clone https://github.com/petereijgermans11/progressive-web-app
```

Ga vervolgens in je terminal naar de directory: `cd pwa-article/pwa-app-manifest-init`. Installeer de dependencies middels: `npm i && npm start` en open de webapp op: `http://localhost:8080`. Zie afbeelding 1.

### Uit wat voor technische componenten bestaat een PWA?

PWA heeft drie belangrijke technische componenten die samenwerken en de web-app activeren. De volgende componenten zijn vereist om een goede PWA te ontwikkelen: De

Manifest-file, de Service Worker en de PWA moet draaien onder https.

### Begrip van de App Manifest file

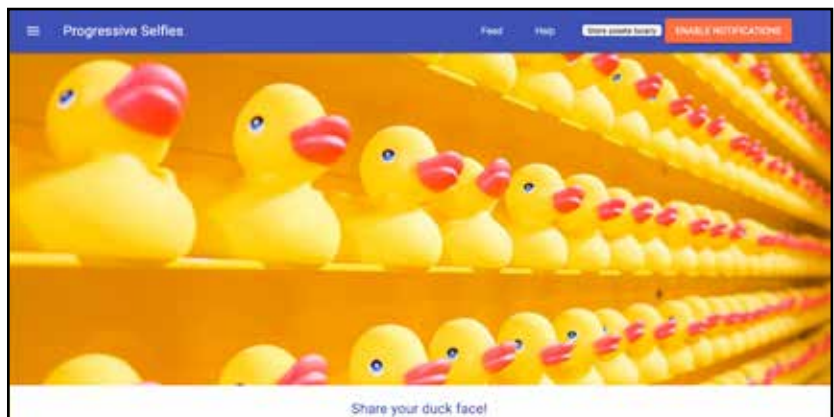
De Manifest file is een configuratie-JSON-bestand dat de informatie van je PWA bevat, zoals het icon dat op het startscherm wordt weergegeven wanneer het is geïnstalleerd, de korte naam van de webapp of de achtergrondkleur. Als de Manifest file aanwezig is, activeert de Chrome-browser automatisch de banner voor het installeren van de web-app (de 'Add to Home Screen' button). Als de gebruiker hiermee instemt, wordt het icon aan het startscherm toegevoegd en wordt de PWA geïnstalleerd.

### Aanmaken van de Manifest.json

Een minimaal Manifest.json bestand voor een PWA ziet er als volgt uit (zie Listing 1).



**Peter Eijgermans** is Frontend Developer en Codesmith bij Ordina JTech. Hij deelt graag zijn kennis met anderen middels workshops en presentaties.



Afbeelding 1.

## Vertel de browser over je Manifest

Maak een Manifest.json bestand aan op hetzelfde niveau als de index.html.

Wanneer je het Manifest hebt gemaakt, voeg je een *link tag* toe aan je index.html (bij de overige *links tags*).

```
<link rel="manifest" href="/manifest.json">
```

## Belangrijke Manifest properties

Je moet ten minste de **property short\_name** of **name** opgeven. De `short_name` wordt gebruikt op het home screen van de gebruiker. *Name* wordt gebruikt in de app-install prompt. Wanneer een gebruiker je PWA toevoegt aan zijn home screen, kun je een set **icons** definiëren die de browser moet gebruiken. Deze icons worden gebruikt op plaatsen zoals het home screen en de app launcher. De **start\_url property** vertelt de browser vanaf waar deze de applicatie moet opstarten. De **scope property** definieert de set URL's die de browser als onderdeel van je app beschouwt en wordt gebruikt om te beslissen wanneer de gebruiker de app heeft verlaten en teruggestuurd moet worden naar een browsertabblad. Je **start\_url** moet zich binnen de scope bevinden. De **display property** geeft de volgende display mode aan:

**Fullscreen:** alle beschikbare ruimte wordt gebruikt voor de app.

**Standalone:** de applicatie heeft de look and feel als een standalone applicatie.

De **background\_color property** wordt gebruikt op de 'splash screen' wanneer de applicatie opgestart is.

Vanaf dit punt werkt de button 'Add to Home Screen' nog niet. Maar je kan wel alvast experimenteren met de Manifest file. Er zijn momenteel diverse tools waarmee je je Manifest file kan genereren, zoals <https://app-Manifest.firebaseapp.com/>

- Genereer je eigen Manifest en plaats deze op hetzelfde niveau als de index.html.
- Check nu in je Chrome Developer Tools of je Manifest actief is. Klik met je *rechtermuisk* op de startpagina van de *Progressive Selfies App* en selecteer 'Inspect'. Kies het tabblad 'Application' en kies 'Manifest'. Restart eventueel de server met **npm start**.

## Wat is een Service Worker?

Een *Service Worker* (SW) is slechts een stukje JavaScript die werkt als een proxy tussen

```
{
  "name": "Progressive Selfies",
  "short_name": "PWA Selfies",
  "icons": [
    {
      "src": "/src/images/icons/app-icon-192x192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/src/images/icons/app-icon-512x512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/index.html",
  "scope": ".",
  "display": "standalone",
  "background_color": "#fff"
}
```

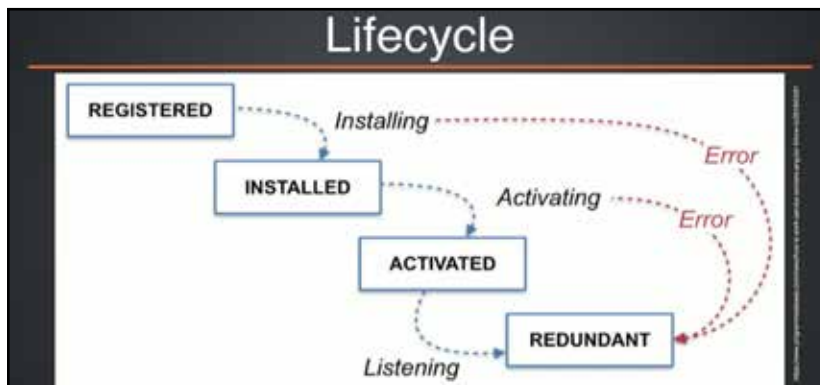
Listing 1

de browser en het netwerk. Een SW ondersteunt o.a. *push notifications*, *background sync*, *caching ect*. De kernfunctie van een SW is de mogelijkheid om netwerk requests te onderscheppen, af te handelen, inclusief het beheren van een cache met responses. De reden dat dit zo'n spannende API is, is dat het je in staat stelt om offline te kunnen blijven werken door o.a. gebruik te maken van de caching.

## De Service Worker Lifecycle

Bij Service Workers worden de volgende stappen in acht genomen voor de basisinstelling:

- Registreren van je SW. Om een SW voor je site te installeren, moet je deze eerst registreren. Als de SW geregistreerd is, start de browser automatisch de installatie obv een *install-event*.
- Wanneer de SW is geïnstalleerd, ontvangt deze een *activate-event*. Deze activate-event kan gebruikt worden voor het opruimen van bronnen die in eerdere versies van een SW zijn gebruikt (zie afbeelding 2).



Afbeelding 2.

Voor de SW, maak een *lege file* met de naam **sw.js** op hetzelfde niveau als `index.html`. En in de **index.html** voeg je een `base` tag toe (bij de overige *links tags in de <head> - sectie*). Dit is de base URL voor al je relatieve links in je app:

```
<base href="/" />
```

Voeg tenslotte onderstaande code toe in **src/js/app.js** om de SW te registreren. Deze code gaat af tijdens het 'laden' van de eerste pagina (zie Listing 2).

Deze code controleert of de API van de SW beschikbaar is in de **navigator property** van het *window - object*. Het window object representeert de browser window (Javascript en ook de *navigator property* is onderdeel van het window object). Als de SW beschikbaar is in de navigator, wordt de SW *geregistreerd* zodra de pagina is geladen. Je kunt nu controleren of een SW is ingeschakeld in de Chrome Developer Tools in tabblad *Application* -> *Service Workers*. Refresh de pagina hiervoor!

## Waarom kan mijn Service Worker zich niet registreren?

Dat kan de volgende oorzaken hebben:

- Dit komt omdat je app niet draait onder HTTPS. Tijdens het ontwikkelen kan je de SW gebruiken via *localhost*. Maar als je het deployed op een site dan heb je een HTTPS setup nodig.
- Het pad van de SW is niet correct. **Update on reload** moet aangevinkt zijn tijdens development! (Zie afbeelding 3.)

## Service Worker Events

Naast **install** en **activate** hebben we een **message** event. Dit event vindt plaats als er een message is ontvangen vanuit een ander script in de App. Het **fetch** event wordt getriggerd wanneer een pagina van je site een netwerkbron vereist. Het kan een nieuwe pagina zijn, een JSON API, een image, een CSS-bestand ect.

Het **sync** event wordt verzonden wanneer er

```

window.addEventListener('load', () => {
  const base = document.querySelector('base');
  let baseUrl = base && base.href || '';
  if (!baseUrl.endsWith('/')) {
    baseUrl = `${baseUrl}/`;
  }

  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register(`${baseUrl}sw.js`)
      .then( registration => {
        // Registration was successful
        console.log('ServiceWorker registration successful with
scope: ', registration.scope);
      })
      .catch(err => {
        // registration failed :(
        console.log('ServiceWorker registration failed: ', err);
      });
  }
});

```

Listing 2.

```

self.addEventListener('install', event => {
  console.log('[Service Worker] Installing Service Worker ...',
event);
  event.waitUntil(self.skipWaiting());
});

self.addEventListener('activate', event => {
  console.log('[Service Worker] Activating Service Worker ...',
event);
  return self.clients.claim();
});

```

Listing 3.

```

self.addEventListener('fetch', event => {
  console.log('[Service Worker] Fetching something ....', event);

  // This fixes a weird bug in Chrome when you open the Developer
Tools
  if (event.request.cache === 'only-if-cached' && event.request.
mode !== 'same-origin') {
    return;
  }

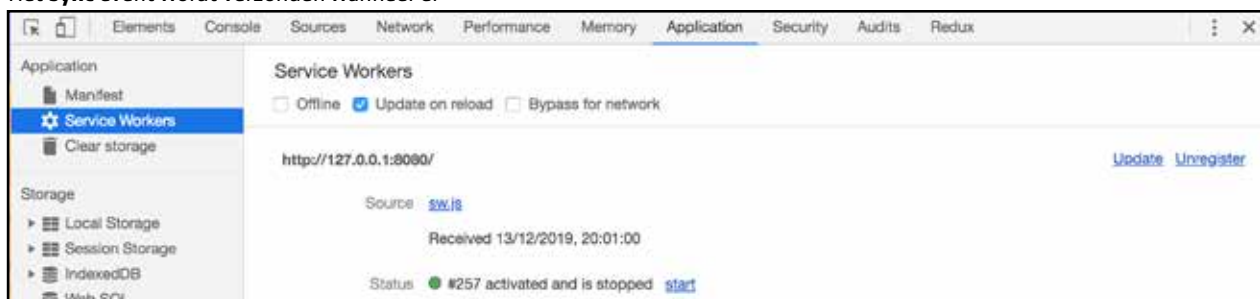
  event.respondWith(fetch(event.request));
});

```

Listing 4.

na een netwerk onbeschikbaarheid weer een netwerk verbinding is. En het **push** event wordt aangeroepen door de Push API van je

Afbeelding 3.



browser als er een push message ontvangen wordt vanuit de backend.

Voeg de volgende code toe in je SW om te luisteren naar de lifecycle events: **install** en **activate** (zie listing 3):

Het *install* event roept de functie *self.skipWaiting()* aan om vervolgens het *activate* event direct te activeren. De *self.clients.claim()* functie in de *activate* event zorgt ervoor dat updates in de SW direct effect hebben. In deze context representeert de **self**-property het window-object (dus je browser window).

## Add to Home Screen

Middels de 'Add to Home Screen-button' (afbeelding 3.) kan een gebruiker de PWA installeren op zijn device. Om de PWA daadwerkelijk te kunnen installeren met deze button moet je in de SW een **fetch event handler** definiëren. Laten we dat even fixen in de **sw.js** (listing 4).

Vink **Update on reload** aan en **Unregister** je SW eerst in Chrome Dev tools en refresh je scherm. Ga naar je PWA en click de '*Customize button*' in Chrome en selecteer: **Install Progressive Selfies...** Hierna wordt de 'install banner' getoond (zie afbeelding 4).

## Service Worker Caching

De kracht van Service Workers ligt in hun vermogen om HTTP-requests te onderscheppen. In deze stap gebruiken we deze mogelijkheid om HTTP-requests en responses te onderscheppen om gebruikers een bliksemsnelle response rechtstreeks vanuit de *cache* te bieden.

## Precaching gedurende Service Worker installation

Wanneer de gebruiker de website voor het eerst bezoekt, begint de SW zichzelf te installeren. Tijdens deze installatiefase kun je de *cache* vullen met alle pagina's, scripts en styling files die de PWA gebruikt. Vul de **sw.js** file als volgt aan (zie listing 5): Deze code maakt gebruik van het **install-event** en voegt in deze fase een array met bestanden `URLS_TO_PRECACHE` toe. Je kunt zien dat zodra de cache is geopend (*caches.open*), je vervolgens bestanden kunt toevoegen middels de *cache.addAll()*. De methode **event.waitUntil()** maakt gebruik van een JavaScript-promise (een vorm van asynchrone communicatie) om te weten of het cachen is

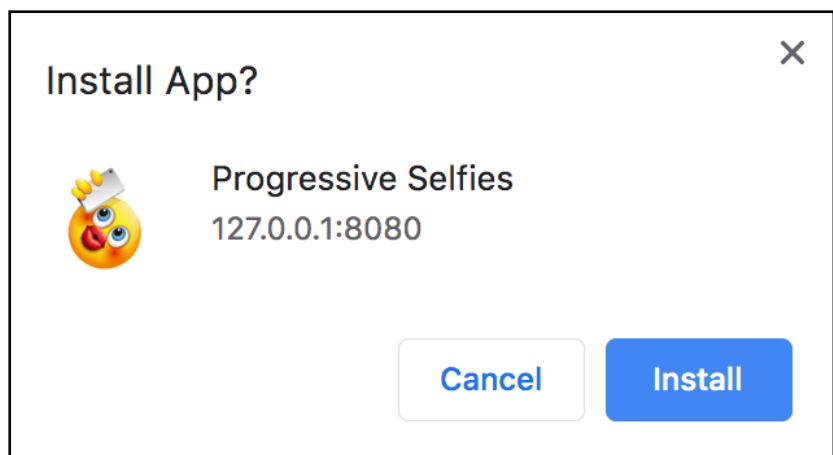
```
const CACHE_STATIC_NAME = 'static';
const URLS_TO_PRECACHE = [
  '/',
  'index.html',
  'src/js/app.js',
  'src/js/feed.js',
  'src/lib/material.min.js',
  'src/css/app.css',
  'src/css/feed.css',
  'src/images/main-image.jpg',
  'https://fonts.googleapis.com/css?family=Roboto:400,700',
  'https://fonts.googleapis.com/icon?family=Material+Icons',
];

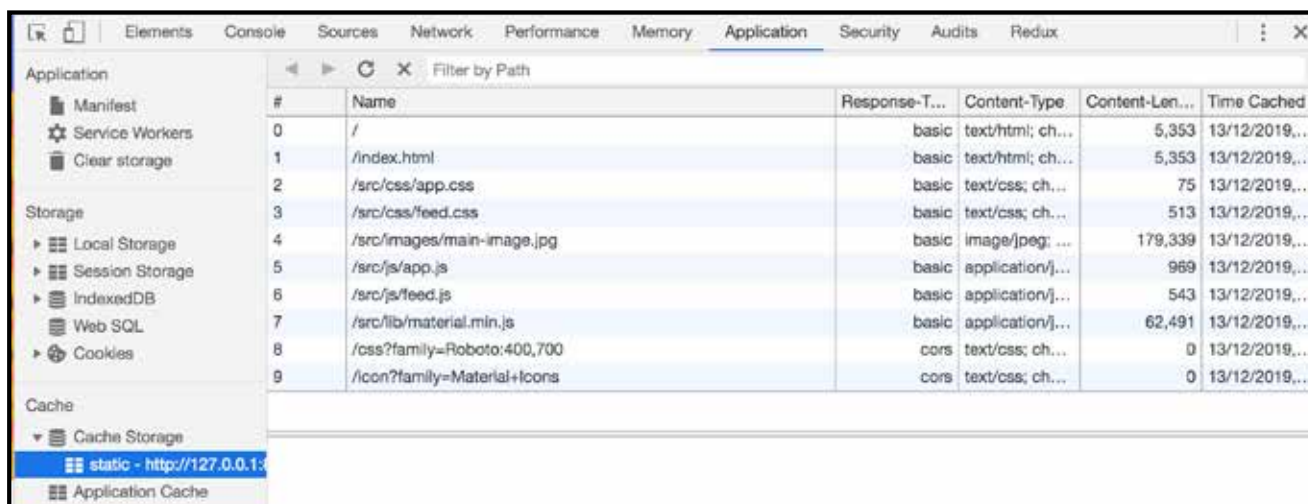
self.addEventListener('install', event => {
  console.log('[Service Worker] Installing Service Worker ...',
    event);
  event.waitUntil(
    caches.open(CACHE_STATIC_NAME)
      .then(cache => {
        console.log('[Service Worker] Precaching App Shell');
        cache.addAll(URLS_TO_PRECACHE);
      })
      .then(() => {
        console.log('[ServiceWorker] Skip waiting on in-
stall');
        return self.skipWaiting();
      })
  );
});
```

Listing 5.

gelukt. Het *install* event roept de functie *self.skipWaiting()* aan om de SW direct te activeren. Als alle bestanden met succes in de cache zijn opgeslagen, wordt de SW geïnstalleerd. Als één van de bestanden niet kan worden gedownload, mislukt de installatie stap. In de Chrome Developer Tools kan je checken of de cache (in de Cache Storage) is gevuld met de statische bestanden uit de `URLS_TO_PRECACHE`-array (zie afbeelding 5).

Afbeelding 4.





#	Name	Response-T...	Content-Type	Content-Len...	Time Cached
0	/	basic	text/html; ch...	5,353	13/12/2019,...
1	/index.html	basic	text/html; ch...	5,353	13/12/2019,...
2	/src/css/app.css	basic	text/css; ch...	75	13/12/2019,...
3	/src/css/feed.css	basic	text/css; ch...	513	13/12/2019,...
4	/src/images/main-image.jpg	basic	image/jpeg; ...	179,339	13/12/2019,...
5	/src/js/app.js	basic	application/j...	969	13/12/2019,...
6	/src/js/feed.js	basic	application/j...	543	13/12/2019,...
7	/src/lib/material.min.js	basic	application/j...	62,491	13/12/2019,...
8	/css?family=Roboto:400,700	cors	text/css; ch...	0	13/12/2019,...
9	/icon?family=Material+Icons	cors	text/css; ch...	0	13/12/2019,...

Afbeelding 5.

Maar als je op het **Netwerk** tabblad kijkt, worden de bestanden nog steeds opgehaald via het netwerk. De reden is dat de cache klaar is voor gebruik, maar we lezen er nog niets uit. Om dat te doen, moeten we de volgende code toevoegen aan onze SW. We beginnen met het luisteren naar het **fetch**-event (zie Listing 6).

We controleren of de inkomende URL (event. request) overeenkomt met iets dat in onze huidige cache zou kunnen voorkomen met behulp van de functie **caches.match()**. Als dit het geval is, retourneer je de gevraagde pagina (response) uit de cache, zo niet dan haal je de gevraagde pagina op middels een netwerk-request: **fetch(event.request)**. Op het moment dat we de checkboxes **Offline** en **Update on reload** aanvinken in het *Application* tabblad en de pagina refreshen, zien we de gecachte versie van onze app (afbeelding 1).

## Background fetch

*Background fetch* API is een SW background feature die het mogelijk maakt om

grote bestanden, films, podcasts ect. op de achtergrond te downloaden. Gedurende de fetch/transfer kan je gebruiker ervoor kiezen om het tabblad te sluiten of zelfs de gehele browser te sluiten. Dit zal de 'transfer' niet stoppen. Nadat de browser weer geopend is, wordt de transfer weer hervat. Ook kan deze API omgaan met slechte bereikbaarheid. De voortgang van de transfer kan getoond worden aan de gebruiker en deze kan het proces

```
self.addEventListener('fetch', event => {
  console.log('[Service Worker] Fetching something ....', event);

  event.respondWith(
    caches.match(event.request)
      .then(response => {
        if (response) {
          console.log(response);
          return response;
        }

        return fetch(event.request);
      })
  );
});
```

Listing 6.

```
window.addEventListener('load', () => {
  ...
  bgFetchButton = document.querySelector('#bgFetchButton');
  bgFetchButton.addEventListener('click', async event => {
    try {
      const registration = await navigator.serviceWorker.ready;
      registration.backgroundFetch.fetch('my-fetch', [new Request(`${baseUrl}src/images/main-image-lg.jpg`)]);
    } catch (err) {
      console.error(err);
    }
  });
  ...
});
```

Listing 7.

```

self.addEventListener('backgroundfetchsuccess', event => {
  console.log('[Service Worker]: Background Fetch Success', event.registration);
  event.waitUntil(
    (async function() {
      try {
        // Iterating the records to populate the cache
        const cache = await caches.open(event.registration.id);
        const records = await event.registration.matchAll();
        const promises = records.map(async record => {
          const response = await record.responseReady;
          await cache.put(record.request, response);
        });
        await Promise.all(promises);
      } catch (err) {
        console.log('[Service Worker]: Caching error');
      }
    })()
  );
});

```

Listing 8.

eventueel pauzeren of afbreken. En tenslotte heeft je PWA toegang tot de data / bronnen die zijn opgehaald.

## Experimental Web Platform features

Background fetch werkt alleen als je 'Experimental Web Platform features' aanzet hebt via de url: **chrome://flags/** Hieronder volgt een voorbeeld hoe je zo'n *Background fetch* implementeert.

Voeg deze button met id 'bgFetchButton' toe in je *index.html* (tussen de andere buttons in de header).

```
<button id="bgFetchButton">Store assets locally</button>
```

En voeg de vet gedrukte code voor het uitvoeren van een *Background fetch* toe in je **app.js** in de **load-event handler** (zie Listing 7).

Bovenstaande code voert een *Background fetch* uit onder de volgende condities:

- De gebruiker klikt op de button met id bg-FetchButton (de onclick-event gaat dan af)
- De SW moet geregistreerd zijn.

Deze check en de *Background fetch* vindt plaats binnen een *async-function* omdat dit proces asynchroon uitgevoerd moet worden zonder de gebruiker te blokkeren.

Vul de cache in *sw.js* (zie listing 8).

Deze code bestaat uit de volgende stappen:

- Zodra het ophalen van de *Background fetch* is voltooid, ontvangt je SW de **background-fetchsuccess-event**

- Creëer en open een nieuwe cache met dezelfde naam als de *registration.id*
- Verkrijg alle records middels *registration.matchAll()*
- Bouw een array op een asynchrone wijze met promises door de records te doorlopen. Wachten tot de records met responses gereed zijn en daarna deze responses in de cache op te slaan met *cache.put()* (zie afbeelding 6.), de *Cache Storage*.
- En tenslotte het uitvoeren van al de promises, middels *Promise.all()*.

## Tenslotte

Na deze introductie kun je verder gaan met een uitgebreide tutorial die je kan vinden in: <https://github.com/petereijgermans11/progressive-web-app/tree/master/pwa-workshop>. Deze tutorial gaat dieper in op zaken als het genereren van je SW middels Workbox, Caching strategieën, Web Push Notifications, Background synchronisatie met een IndexedDB-API en diverse andere nieuwe Web-API's. ■

Afbeelding 6.

