

PWA

Deel 2: Bouw zelf een Progressive Selfies Web App met JavaScript

Als vervolg op het eerste artikel over PWA, gaan we ons richten op het inzetten van API's die toegang hebben tot de hardware op je device. Omgevingslichtsensoren, accelerometers en 'face and shape detectie' zijn voorbeelden van API's die momenteel worden ontwikkeld.

Als je een krachtige PWA wilt bouwen die gebruik maakt van de hardware op een device, wordt het alleen maar beter.

In dit artikel ga ik in op enkele PWA-features die toegang geven tot je hardware API's:

- **Media Capture API**, om een foto (in dit artikel een 'selfie') te kunnen maken met je camera
https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API
- **GeoLocation API**, om de locatie van je selfie te bepalen
https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API
- en de veelgebruikte **Background Sync API**, om de selfies tussentijds in een wachtrij of database te plaatsen, zodat de gebruiker ook een selfie naar de server kan zenden als deze *geen connectie* heeft.
<https://developers.google.com/web/updates/2015/12/background-sync>

Project Setup

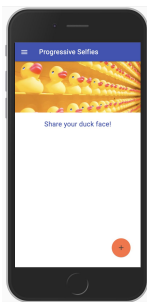
Om te starten dien je eerst *node* te installeren: <https://nodejs.org/en/download/>

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-native-features-init**

Installeer de dependencies middels: **npm i && npm start** en open de webapp op: **http://localhost:8080** (zie afbeelding 1)



Afbeelding 1

Media Capture API Met de Media Capture API hebben webapps toegang tot de streams via de audio- en video-opname-interfaces van je device. De streams die door de API worden weergegeven, kunnen rechtstreeks worden gekoppeld aan de HTML-elementen `<audio>` of `<video>`, of kunnen worden gelezen en gemanipuleerd in de code. Inclusief verdere, meer specifieke verwerking via de Media Recorder API of Real-Time Communication API.

API uitleg

```
navigator.mediaDevices.getUserMedia(constraints)
```

Hiermee wordt de gebruiker gevraagd om toegang tot de media-interface (video of audio).

```
stream.getAudioTracks()
```

Retourneert een verzameling audiotracks die door de microfoon van je device worden geleverd.

```
stream.getVideoTracks()
```

Retourneert een verzameling videotracks die door de camera van je device worden geleverd.

```
mediaElement.srcObject = stream
```

Stelt een stream in die moet worden gerenderd in het meegeleverde <audio> of <video> HTML-element.

De Progressive Selfies-app aanpassen om selfies te kunnen maken

Voeg onderstaande code toe in je **index.html** (zie listing 1), direct onder de tag: **<div id="create-post">**. In deze code wordt de *'Capture button'* gedefinieerd om een snapshot (=selfie) te maken van je videotracks. Tevens is de <video> en <canvas> tag gedefinieerd voor het respectievelijk tonen van je video en je snapshot (selfie) in je webpagina.

```
<video id="player" autoplay></video>
<canvas id="canvas" width="320px" height="240px"></canvas>
<button id="capture-btn"
        class="mdl-button mdl-js-button mdl-button--raised mdl-button--colored">
    Capture
</button>
```

Listing 1

Voeg onderstaande code toe in **feed.js** (listing 2), voor het definiëren van je benodigde variabelen. Hierbij bevat bijvoorbeeld de variabele **videoPlayer** het *HTML-element* <video> met het id = "player". Hierin worden je videotracks gerenderd. Het **canvasElement** is voor het renderen van je selfie, de **captureButton** is er om een selfie te kunnen maken.

```
const videoPlayer = document.querySelector('#player');
const canvasElement = document.querySelector('#canvas');
const captureButton = document.querySelector('#capture-btn');
let picture;
```

Listing 2

Voeg onderstaande **initializeMedia-functie** toe in **feed.js** (listing 3), om je camera te initialiseren:

```
const initializeMedia = () => {
  if (!('mediaDevices' in navigator)) {
    navigator.mediaDevices = {};
  }
  if (!('getUserMedia' in navigator.mediaDevices)) {
    navigator.mediaDevices.getUserMedia = (constraints) => {
      const getUserMedia = navigator.webkitGetUserMedia ||
        navigator.mozGetUserMedia;

      if (!getUserMedia) {
        return Promise.reject(new Error('getUserMedia is not implemented!'));
      }
      return new Promise((resolve, reject) =>
        getUserMedia.call(navigator, constraints, resolve, reject));
    };
  }

  navigator.mediaDevices.getUserMedia({video: {facingMode: 'user'}, audio: false})
    .then(stream => {
      videoPlayer.srcObject = stream;
      videoPlayer.style.display = 'block';
      videoPlayer.setAttribute('autoplay', '');
      videoPlayer.setAttribute('muted', '');
      videoPlayer.setAttribute('playsinline', '');
    })
    .catch(error => {
      console.log(error);
    });
};
```

```
};
```

Listing 3

Deze code initialiseert de camera. Eerst wordt gecontroleerd of de API van de 'mediaDevices' en 'getUserMedia' 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 'mediaDevices' en 'getUserMedia' beschikbaar zijn in de navigator, wordt in bovenstaande code de gebruiker gevraagd om toegang tot de camera, middels de call: **navigator.mediaDevices.getUserMedia(constraints)**.

De call: **videoPlayer.srcObject = stream**, stelt een stream (of videotracks) in, die wordt gerenderd in het meegeleverde <video> HTML-element.

Voeg onderstaande code toe in **feed.js** (listing 4), voor het definiëren van je 'modal' om een selfie maken. Hierin wordt ook de bovenstaande **initializeMedia-functie** aangeroepen.

```
const openCreatePostModal = () => {  
  setTimeout(() => createPostArea.style.transform = 'translateY(0)', 1);  
  initializeMedia();  
};
```



Listing 4

Afbeelding 2

Voeg een *click* event handler toe aan de 'shareImageButton' in **feed.js** (zie listing 5). Deze button (afbeelding 2) *opent* de hierboven gedefinieerde 'openCreatePostModal' om een selfie te maken.

```
shareImageButton.addEventListener('click', openCreatePostModal);
```

Listing 5

Voeg tenslotte een click event handler toe voor 'Capture Button' in **feed.js** (listing 6). Met deze 'Capture Button' kan je een snapshot/selfie maken van je videotracks (zie afbeelding 3). Deze snapshot wordt gerenderd in het *canvasElement* en wordt geconverteerd naar een Blob (zie listing 7) via de functie: *dataURLtoBlob* (voor eventuele opslag in een Database).



Afbeelding 3

```
captureButton.addEventListener('click', event => {
  canvasElement.style.display = 'block';
  videoPlayer.style.display = 'none';
  captureButton.style.display = 'none';
  const context = canvasElement.getContext('2d');
  context.drawImage(
    videoPlayer, 0, 0, canvasElement.width,
    videoPlayer.videoHeight / (videoPlayer.videoWidth / canvasElement.width)
  );
  videoPlayer.srcObject.getVideoTracks().forEach(track => track.stop());
  picture = dataURItoBlob(canvasElement.toDataURL());
});
```

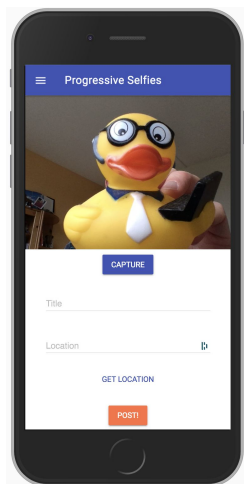
Listing 6

In `utility.js`

```
const dataURItoBlob= dataURI => {
  const byteString = atob(dataURI.split(',')[1]);
  const mimeType = dataURI.split(',')[0].split(':')[1].split(';')[0];
  const ab = new ArrayBuffer(byteString.length);
  const ia = new Uint8Array(ab);
  for (let i = 0; i < byteString.length; i++) {
    ia[i] = byteString.charCodeAt(i);
  }
  const blob = new Blob([ab], {type: mimeType});
  return blob;
};
```

Listing 7

Restart eventueel de server met ***npm start*** en maak een selfie middels de ***Capture button*** (afbeelding 4).



Afbeelding 4

De Progressive Selfies-app aanpassen om je locatie te bepalen

Geolocation

Met de Geolocation API hebben webapplicaties toegang tot de locatiegegevens die door het apparaat worden verstrekt - verkregen via GPS of via de netwerkomgeving. Afgezien van de eenmalige locatie vraag, biedt het ook een manier om de app op de hoogte te stellen van locatie wijzigingen.

API uitleg

navigator.geolocation.getCurrentPosition(callback)

Voert een eenmalige zoekopdracht uit voor de locatie met coördinaten, nauwkeurigheid, hoogte en snelheid, indien beschikbaar.

navigator.geolocation.watchPosition(callback)

Locatie wijzigingen worden hiermee geobserveerd.

Laten we de Geolocation API gebruiken om de positie te bepalen van je selfie.

Voeg onderstaande code toe in je **index.html**, direct onder de tag: **div#manual-location**.

In deze code wordt de 'Get Location button' gedefinieerd om de locatie te bepalen waar je de selfie hebt genomen (listing 8).

```
<div class="input-section">
  <button
    id="location-btn"
    type="button"
    class="mdl-button mdl-js-button mdl-button--colored">
    Get Location
  </button>
  <div
    id="location-loader"
    class="mdl-spinner mdl-js-spinner is-active">
  </div>
</div>
```

Listing 8

Voeg onderstaande code toe in **feed.js**, voor het definiëren van je benodigde variabelen om de locatie te bepalen (listing 9):

```
const locationButton = document.querySelector('#location-btn');
const locationLoader = document.querySelector('#location-loader');
let fetchedLocation = {lat: 0, lng: 0};
```

Listing 9

Voorafgaand aan initializeMedia()-functie in **feed.js** (listing 10):

```
const initializeLocation = () => {
  if (!('geolocation' in navigator)) {
    locationButton.style.display = 'none';
  }
};
```

Listing 10

Voeg de volgende **initializeLocation()**-functie toe in **openCreatePostModal**, direct na initializeMedia() in **feed.js** (zie listing 11);

```
const openCreatePostModal = () => {
  setTimeout(() => createPostArea.style.transform = 'translateY(0)', 1);
```

```
    initializeMedia();
    initializeLocation();
};
```

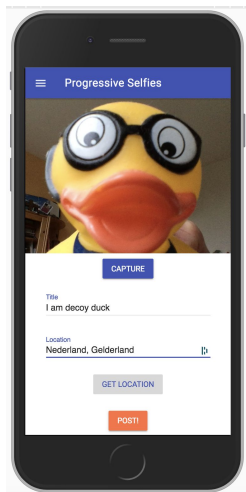
Listing 11

Voeg een click event handler toe voor de **'locationButton'** in **feed.js**. Met deze *'Location button'* wordt de locatie bepaald waar je de selfie hebt genomen (listing 12):

```
locationButton.addEventListener('click', event => {
  if (!('geolocation' in navigator)) {
    return;
  }
  let sawAlert = false;
  locationButton.style.display = 'none';
  locationLoader.style.display = 'block';
  navigator.geolocation.getCurrentPosition(position => {
    locationButton.style.display = 'inline';
    locationLoader.style.display = 'none';
    fetchedLocation = {lat: position.coords.latitude, lng: position.coords.longitude};
    const reverseGeocodeService = 'https://nominatim.openstreetmap.org/reverse';
    fetch(`${reverseGeocodeService}?
      format=jsonv2&lat=${fetchedLocation.lat}&lon=${fetchedLocation.lng}`)
      .then(response => response.json())
      .then(data => {
        locationInput.value = `${data.address.country}, ${data.address.state}`;
        document.querySelector('#manual-location').classList.add('is-focused');
      })
      .catch(error => {
        console.log(error);
        locationButton.style.display = 'inline';
        locationLoader.style.display = 'none';
        if (!sawAlert) {
          alert('Couldn\'t fetch location, please enter manually!');
          sawAlert = true;
        }
        fetchedLocation = {lat: 0, lng: 0};
      });
  }, error => {
    console.log(error);
    locationButton.style.display = 'inline';
    locationLoader.style.display = 'none';
    if (!sawAlert) {
      alert('Couldn\'t fetch location, please enter manually!');
      sawAlert = true;
    }
    fetchedLocation = {lat: 0, lng: 0};
  }, {timeout: 7000});
});
```

Listing 12

Deze code controleert of de API van de 'geolocation' beschikbaar is in de **navigator property** van het *window object*. Als dit het geval is dan voert deze code een eenmalige zoekopdracht uit om de locatie te bepalen (met coördinaten), via de **navigator.geolocation.getCurrentPosition()**. Vervolgens wordt met deze coördinaten via *'openstreetmap'*, het adres erbij gezocht.



Afbeelding 5

Restart eventueel de server met ***npm start*** en haal de locatie op middels de 'GET LOCATION' button (afbeelding 5)

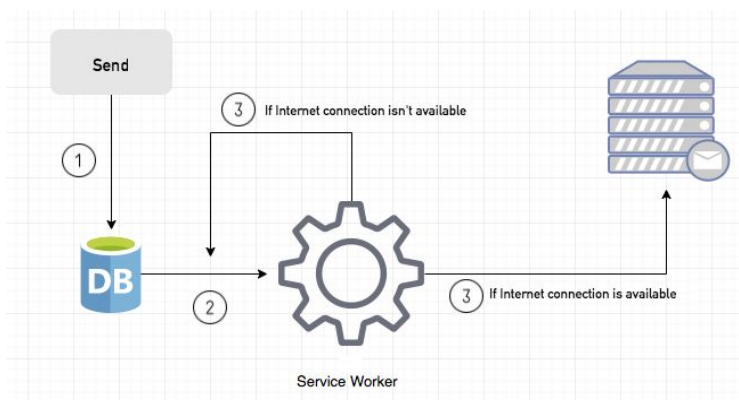
Selfies Online versturen met BackgroundSync API

Wat als je wilt dat de gebruiker gegevens naar de server stuurt terwijl de gebruiker offline is, waardoor dit niet mogelijk is?

Met de BackgroundSync API kunnen gebruikers gegevens in de wachtrij zetten die naar de server moeten worden verzonden, terwijl een gebruiker offline werkt. Zodra deze weer online is, worden de gegevens in de wachtrij naar de server verzonden.

Voorbeeld:

Stel dat iemand die onze Progressive Selfies app gebruikt een selfie wil maken en versturen, maar de app is offline. Met BackgroundSync API kan de gebruiker een selfie in de wachtrij zetten terwijl hij offline is. Zodra deze weer online is, verstuurt de **Service Worker** de gegevens naar de server. In ons geval gebruiken we een **IndexedDB** om gegevens tussentijds op te slaan (afbeelding 6). De library voor deze database kan je vinden in de map **lib/idb.js**. Wat en hoe een **Service Worker** werkt kan je nalezen in mijn eerste artikel over PWA.



Afbeelding 6

Clone de server

Clone eerst de **server** via: <https://github.com/petereijgermans11/progressive-web-app-server>

Naar deze server gaan we je de selfies versturen.

Installeer de dependencies en start de PWA middels: ***npm i && npm start***

De server draait op localhost:3000

Sync selfies

Om **BackgroundSync** op onze app toe te passen, moeten we een 'store' in onze **IndexedDB-database** maken om onze "te synchroniseren selfies" te bewaren (listing 13). We doen dat in **utility.js**. Deze utility.js bevat de code die nodig is voor zowel de **Service Worker** als de app zelf.

```
const dbPromise = idb.openDb('selfies-store', 1, upgradeDB => {
  if (!upgradeDB.objectStoreNames.contains('selfies')) {
    upgradeDB.createObjectStore('selfies', {keyPath: 'id'});
  }
  if (!upgradeDB.objectStoreNames.contains('sync-selfies')) {
    upgradeDB.createObjectStore('sync-selfies', {keyPath: 'id'});
  }
});
```

Listing 13

We dienen de BackgroundSync API in te zetten als we gegevens naar de server zenden, via de **submit** event. Voeg een **submit** event handler toe in **feed.js** (Listing 14):

```
form.addEventListener('submit', event => {
  event.preventDefault();
  if (titleInput.value.trim() === '' || locationInput.value.trim() === '' || !picture) {
    alert('Please enter valid data!');
    return;
  }
  closeCreatePostModal();

  const id = new Date().getTime();

  if ('serviceWorker' in navigator && 'SyncManager' in window) {
    navigator.serviceWorker.ready
      .then(sw => {
        const selfie = {
          id: id,
          title: titleInput.value,
          location: locationInput.value,
          selfie: picture,
        };
        writeData('sync-selfies', selfie)
          .then(() => sw.sync.register('sync-new-selfies'))
          .then(() => {
            const snackbarContainer =
              document.querySelector('#confirmation-toast');
            const data = {message: 'Your Selfie was saved for syncing!'};
            snackbarContainer.MaterialSnackbar.showSnackbar(data);
            readAllData('sync-selfies')
              .then(syncSelfies => {
                updateUI(syncSelfies);
              })
          })
          .catch(function (err) {
            console.log(err);
          });
      })
    };
  });
});
```

Listing 14

In het eerste deel van de code wordt de **id** van de te versturen selfie gegenereerd. Eerst doen we een eenvoudige controle om te zien of de browser Service Worker en SyncManager ondersteunt. Als dit het geval is en de **Service Worker** klaar is, registreer dan een synchronisatie (**sync**) met de tag **sync-new-posts**. Dit is een eenvoudige string die wordt gebruikt om deze **sync-event** te herkennen. Je kunt deze sync-tags beschouwen als eenvoudige labels voor verschillende acties.

Vervolgens slaan we de selfie op in de **IndexedDB** middels **writeData**.

Tenslotte, worden alle opgeslagen selfies uitgelezen middels **readAllData('sync-selfies')** en worden getoond in je PWA via de functie: **updateUI(syncSelfies)** (zie afbeelding 8). Er wordt ook nog een message uitgestuurd: 'Your Selfie was saved for syncing!'

De Service Worker

Om de **Service Worker** correct te laten werken, dient er een **sync event-listener** gedefinieerd te worden in **sw.js** (Listing 15):

```
self.addEventListener('sync', event => {
  console.log('[Service Worker] Background syncing', event);
  if (event.tag === 'sync-new-selfies') {
    console.log('[Service Worker] Syncing new Posts');
    event.waitUntil(
      readAllData('sync-selfies')
        .then(syncSelfies => {
          for (const syncSelfie of syncSelfies) {
            const postData = new FormData();
            postData.append('id', syncSelfie.id);
            postData.append('title', syncSelfie.title);
            postData.append('location', syncSelfie.location);
            postData.append('selfie', syncSelfie.selfie);
            fetch(API_URL, {method: 'POST', body: postData})
              .then(response => {
                console.log('Sent data', response);
                if (response.ok) {
                  response.json()
                    .then(resData => {
                      deleteItemFromData('sync-selfies',
                        parseInt(resData.id));
                    });
                }
              })
            .catch(error =>
              console.log('Error while sending data', error));
          }
        })
    );
  }
});
```

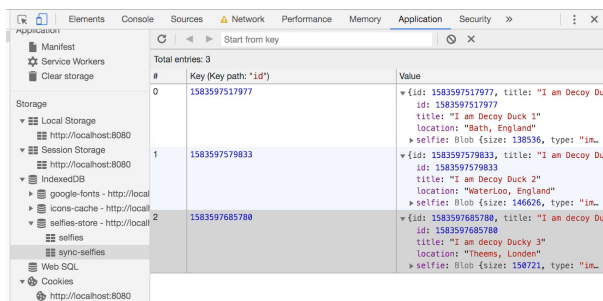
Listing 15

Bovenstaande **sync-event** wordt alleen geactiveerd als de browser weer *verbinding* heeft. De Service Worker kan omgaan met dit soort events (zie mijn eerste artikel over PWA). Er wordt ook gecheckt of het huidige **sync-event** een **tag** heeft die overeenkomt met de string **'sync-new-selfies'**. Als we deze tag niet zouden hebben, wordt het **sync-event** telkens geactiveerd wanneer de gebruiker verbinding heeft. Vervolgens halen we de selfies op die zijn opgeslagen in de **IndexedDB** toen de gebruiker op de submit-button klikte (zie listing 14). Hierna wordt de **fetch-API** gebruikt om de selfies naar de server te verzenden, middels de API_URL: <http://localhost:3000/selfies>. Tenslotte worden de reeds verstuurde selfies verwijderd uit de **IndexedDB**.

Testing

Geloof het of niet, dit alles testen is eenvoudiger dan je denkt: als je de pagina eenmaal hebt bezocht en de **Service Worker** actief is, hoeft je alleen maar de verbinding met het netwerk te verbreken.

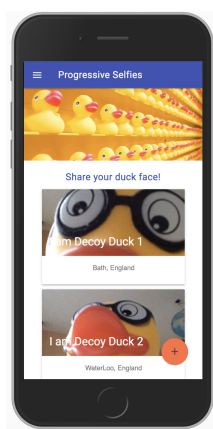
Iedere Selfie die je offline probeert te versturen zal nu opgeslagen worden in de **IndexedDB** (zie afbeelding 7).
Tevens zullen alle opgeslagen selfies getoond worden in je hoofdscherm (zie afbeelding 8).



The screenshot shows the Chrome DevTools Application tab with the IndexedDB database selected. It displays three entries in the IndexedDB database, each representing a selfie record.

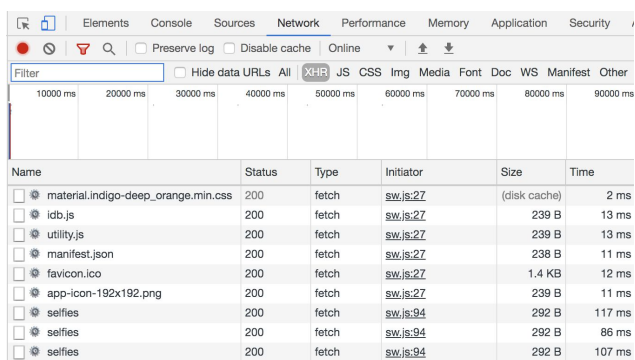
#	Key (Key path: "id")	Value
0	1583597517977	{id: 1583597517977, title: "I am Decoy Duck 1", location: "Bath, England", selfie: Blob (size: 138536, type: "in...")}
1	1583597579833	{id: 1583597579833, title: "I am Decoy Duck 2", location: "Waterloo, England", selfie: Blob (size: 146626, type: "in...")}
2	1583597685780	{id: 1583597685780, title: "I am decoy Duck 3", location: "Theems, London", selfie: Blob (size: 158721, type: "in...")}

Afbeelding 7



Afbeelding 8

Zodra je weer **online** bent zal de **Service Worker** ervoor zorgen dat de selfies verstuurd worden naar de server (bekijk hiervoor je **Network-tab** in je Chrome Developer Tools in afbeelding 9). Daar zie je dat er 3 maal een selfie 'ge-post' is naar de server. Deze server bevat een folder met de naam *'images'*, waar de verstuurde selfies in staan.



The screenshot shows the Chrome DevTools Network tab with a list of network requests. The requests are filtered by 'All' and show various resources loaded by the browser. The last three requests are 'selfies' files, which are the ones being sent to the server.

Name	Status	Type	Initiator	Size	Time
material.indigo-deep_orange.min.css	200	fetch	sw.js:27	(disk cache)	2 ms
icb.js	200	fetch	sw.js:27	239 B	13 ms
utility.js	200	fetch	sw.js:27	239 B	13 ms
manifest.json	200	fetch	sw.js:27	238 B	11 ms
favicon.ico	200	fetch	sw.js:27	1,4 KB	12 ms
app-icon-192x192.png	200	fetch	sw.js:27	239 B	11 ms
selfies	200	fetch	sw.js:94	292 B	117 ms
selfies	200	fetch	sw.js:94	292 B	86 ms
selfies	200	fetch	sw.js:94	292 B	107 ms

Afbeelding 9

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>. In een volgend artikel ga ik verder in op andere API's als Web Streams API, Push API (voor het ontvangen van push notifications) en de web Bluetooth API.

BIO:

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

