



DALL·E 3

PWAs

Progressive Web Apps – Was sie können und wieso PWAs cool sind!



DEMO

Was bauen wir heute?

„APPs“

- Bieten uns Nutzern Funktionalität
- Machen unsere Geräte erst „nutzbar“



„WEB“-APP

- „Anwendungen“ die im „Web“ laufen
 - Interaktives
 - Zugang über den Browser



Gmail



GitLab



Google Docs

„PROGRESSIVE“-WEB-APP

- Klassische „Apps“ sind „nativ“ installiert und ausführbar
 - Leben direkt „auf dem Gerät“
- „Web“-Apps leben „im Browser“
 - „flüchtig“
 - Internetpflicht
- „Progressive“-Web-Apps können installiert werden!
 - Sprung „aus dem Browser“ und „auf das Gerät“
 - Look & Feel einer „normalen“ App
 - Zugriff auf einige „native“ APIs des Geräts
 - Können offline funktionieren!

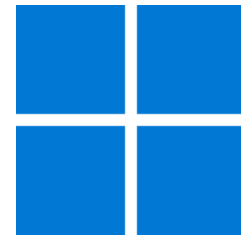


apt-get

PWA

DIE IDEE HINTER PWAs

1. Die Nutzer wollen Apps
2. Wenn wir eine App anbieten, dann sollte diese auf jeder Plattform verfügbar sein
3. Es gibt sehr viele Plattformen
4. Es gibt doch eine Plattform die bereits fast überall läuft: das „Web“



5. ???
6. Profit




~ so oder so ähnlich Steven Champeon in „Progressive Enhancement“ 2003 und Google

PWAs in 2024


- „Basics“ unterstützt in jedem aktuellen Browser
 - Installation
 - Natives „Look & Feel“
 - Offline Fähigkeit
- Einige Features noch Chromium exklusiv

- *Push ist es nicht mehr!*


Push API - WD


 **Baseline 2023** Newly available across major browsers

API to allow messages to be pushed from a server to a browser, even when the site isn't focused or even open in the browser.


manifest: display 

Usage % of all users ?
Global 73.82% + 16.3% = 90.12%


Current aligned Usage relative Date relative Filtered All 


Chrome	Edge *	Safari	Firefox	Opera	IE  *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser
4-38	12-18	3.1-16.6					3.2-11.2							
39-125	79-125	17.0-17.4	2-126	10-110	6-10		11.3-17.4	4-24		12-12.1		2.1-4.4.4		
126	126	17.5	127	111	11	126	17.5	25	all	80	15.5	126	127	14.9
127-129		17.6-TP	128-130				17.6-18.0							

caniuse.com/mdn-html_manifest_display

ServiceWorker API 

Usage % of all users ?
Global 96.1%

Current aligned Usage relative Date relative Filtered All 

Chrome	Edge *	Safari	Firefox	Opera	IE  *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser
4-39	12-16	3.1-11	2-43	10-26			3.2-11.2							
40-125	17-125	11.1-17.4	44-126	27-110	6-10		11.3-17.4	4-24		12-12.1		2.1-4.4.4		
126	126	17.5	127	111	11	126	17.5	25	all	80	15.5	126	127	14.9
127-129		17.6-TP	128-130				17.6-18.0							

caniuse.com/?search=ServiceWorker

AUFGABE 1 - ZÜCKT DIE GABELN!

Wir hosten eine einfache Demo Web-App via GitHub Pages, auf die wir im Verlauf des Workshops aufbauen werden.

1. Forkt euch <https://github.com/OliverWich/WebTech2024-PWA-Workshop>
 2. Richtet ein GitHub Pages Deployment ein
- Details und Tipps findet ihr in der README.md im Ordner „Aufgabe 1“

Ihr habt hierfür ca. 10 Minuten

UND JETZT?

- Wie sorgen wir dafür das die App installiert werden kann?
- Ein „*Web Application Manifest*“
 - .json oder .webmanifest Datei mit technischen Informationen zu unserer App

```
{
  "id": "oliverwich.demo.PWA",
  "name": "Demo PWA",
  "short_name": "PWA",
  "start_url": "https://oliverwich.github.io/WebTech2024-PWA-Workshop/",
  "scope": "https://oliverwich.github.io/WebTech2024-PWA-Workshop/",
  "theme_color": "#2c28f0",
  "background_color": "#ffffff",
  "lang": "de-DE",
  "orientation": "portrait",
  "display": "standalone",
  "icons": [
    {
      "purpose": "maskable",
      "sizes": "512x512",
      "src": "icon512_maskable.png",
      "type": "image/png"
    },
    {
      "purpose": "any",
      "sizes": "512x512",
      "src": "icon512_rounded.png",
      "type": "image/png"
    }
  ]
}
```

DIE WICHTIGSTEN FELDER DES WEB-APP-MANIFESTS

```
{
  "name": "Demo PWA",
  "short_name": "PWA",
  "start_url": "https://oliverwich.github.io/WebTech2024-PWA-Workshop/",
  "display": "standalone",
  "icons": [
    {
      "purpose": "maskable",
      
    },
    {
      "purpose": "any",
      
    }
  ]
}
```

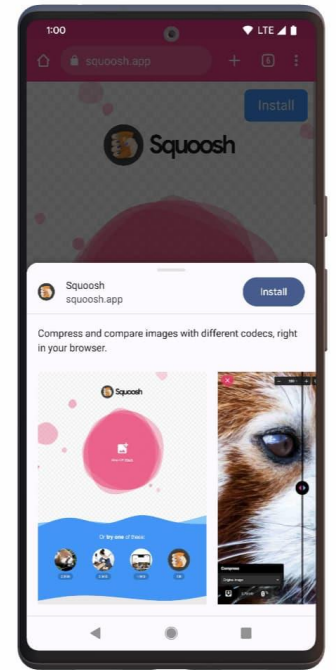
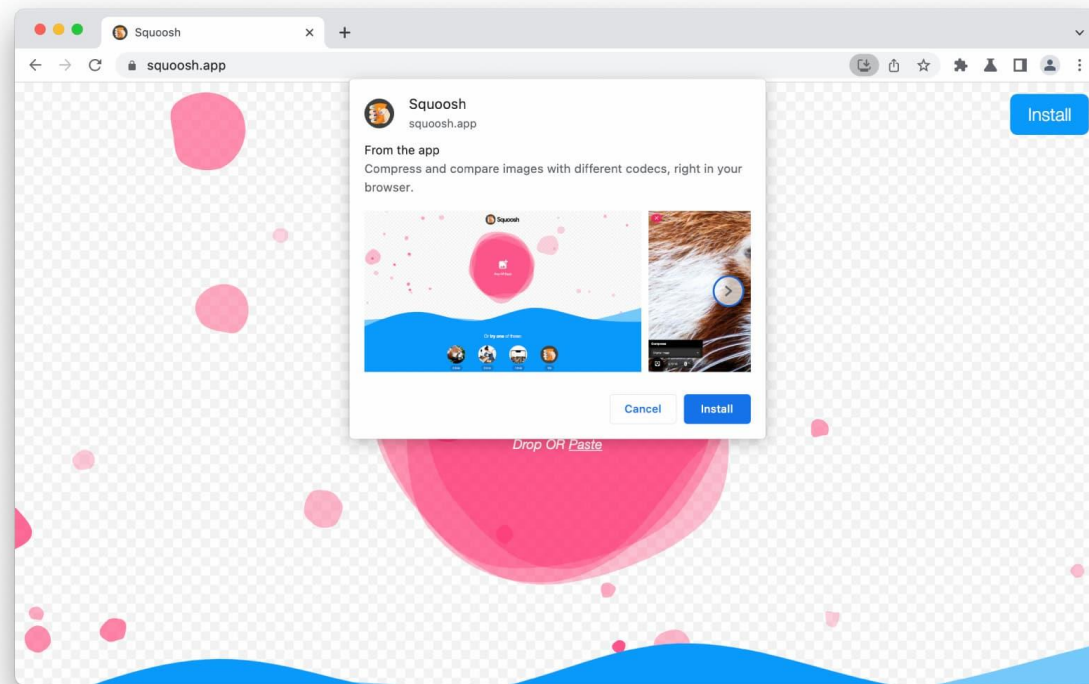
- **"name"**: Der vollständige Name der App
- **"short_name"**: Der Name, der in der App-Übersicht angezeigt wird
- **"start_url"**: Die Startseite der App
- **"display"**: Der gewünschte Display-mode. "standalone" für ein natives Look & Feel
- **"icons"**: Eine Liste mit Icons für unterschiedliche Zwecke

WEITERE FELDER

```
{
  "id": "<eine.einzige.ID>", // Die eindeutige ID der App
  "name": "Demo PWA", // Der volle Name der App
  "short_name": "PWA", // Der Name der App auf dem Homescreen
  "start_url": "<valider start URL>", // Die Start-URL der App
  "theme_color": "#2c28f0", // Die Farbe der Adressleiste (unter Android)
  "background_color": "#ffffff", // Die Farbe des Hintergrunds vom Splashscreen (unter Android)
  "lang": "de-DE", // Die Sprache der App
  "orientation": "portrait", // Die Ausrichtung der App
  "display": "standalone", // "standalone" bedeutet hier das die App wie eine native App funktioniert, ohne Browser-Elemente
  "icons": [
    {
      "purpose": "maskable", // Das Icon für iOS (oder einen Android-Launcher der maskable Icons benutzt)
      "sizes": "512x512",
      "src": "icon512_maskable.png",
      "type": "image/png"
    },
    {
      "purpose": "any", // Das Standard-Icon, falls der Launcher kein maskable Icon unterstützt. Wird von Chrome Desktop benutzt.
      "sizes": "512x512",
      "src": "icon512_rounded.png",
      "type": "image/png"
    }
  ]
}
```

RICH INSTALL UI

- App Vorschau, ähnlich wie in Stores
- Wenn „**screenshots**“ und „**description**“ definiert ist
- Chrome exklusiv 🙄(ツ)🙄



web.dev/patterns/web-apps/richer-install-ui?hl=de

AUFGABE 2 - WEB-APP

Wir machen unser GitHub Pages Deployment installierbar.

1. Schaut euch das Beispiel Manifest im Ordner „Aufgabe 2“ an
 2. Passt es an und benutzt es in eurer Anwendung
 3. Installiert die Web-App auf eurem Smartphone
- Infos und Tipps in der README.md

Ihr habt hierfür ca. **10 Minuten**, falls ihr schnell seid, gibt es eine Bonus Aufgabe!

AUFGABE 3 - EIGENES INSTALLATIONS HANDLING

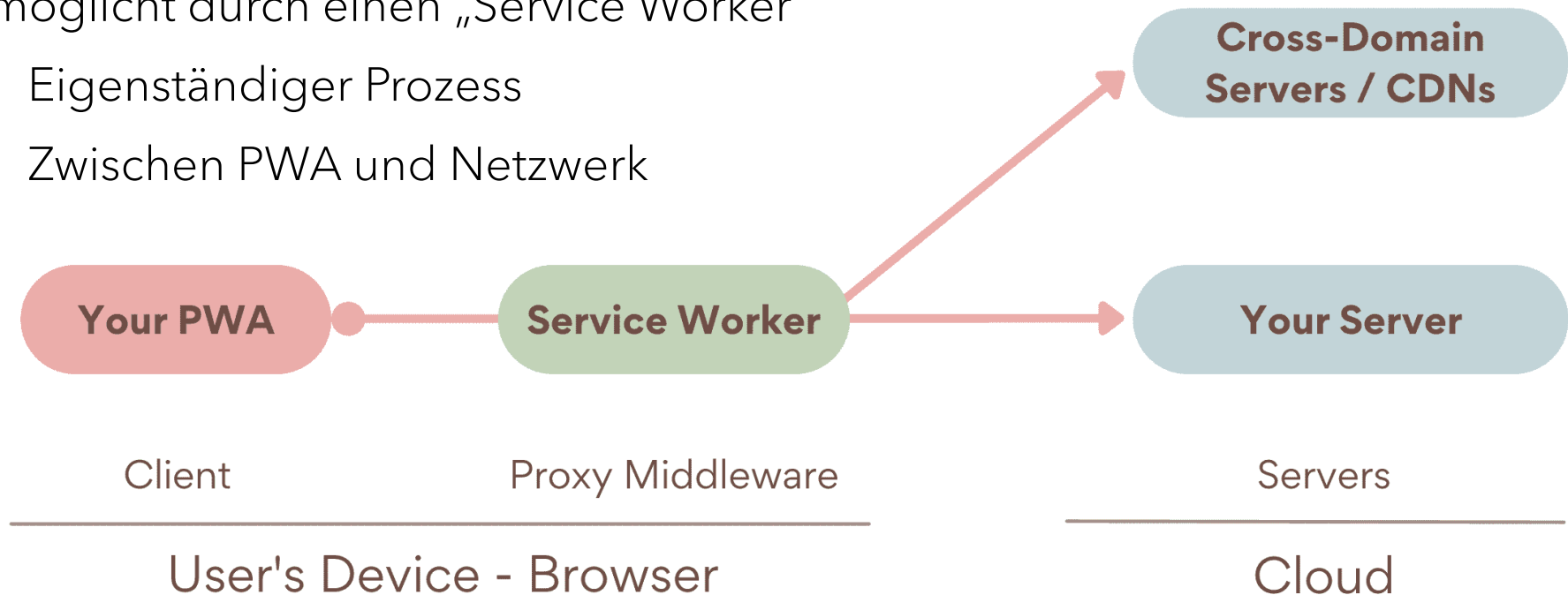
Wir fügen einen FAB („floating action button“) unserer App hinzu, welcher es dem Nutzer ermöglicht, unsere App direkt zu installieren.

1. Bindet FAB.css und den Code aus FAB.html sowie FAB.js in eure Anwendung ein
 2. Implementiert einen „beforeinstallprompt“-Event-Listener welcher den FAB einblendet sobald der Browser das Event ausgelöst hat
- Infos und Tipps natürlich wieder in der README.md im Ordner „Aufgabe 3“

Ihr habt hierfür **bis 14:00** Zeit, oder ihr nehmt euch bis dahin Pause 😊

OFFLINE FUNKTIONALITÄT

- Ermöglicht durch einen „Service Worker“
 - Eigenständiger Prozess
 - Zwischen PWA und Netzwerk



web.dev/learn/pwa/service-workers

SERVICE WORKER LIFECYCLE

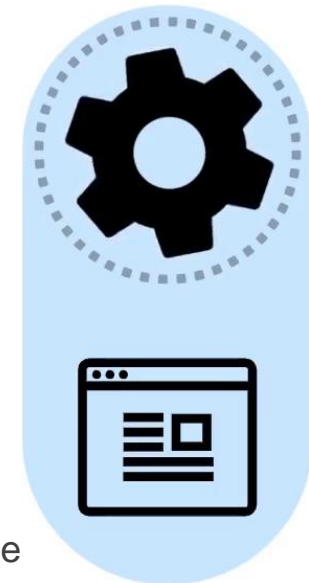
- Wird einmal „Registriert“
- Zwei Schritte für den ersten Service Worker:
 - Installing („**install**“)
 - Active („**activate**“)
- Danach unabhängig davon, ob die App im Vordergrund ist oder nicht

```
if ('serviceWorker' in navigator) {  
  navigator.serviceWorker.register(  
    scriptURL: './serviceWorker.js'  
  )  
}
```

Installing



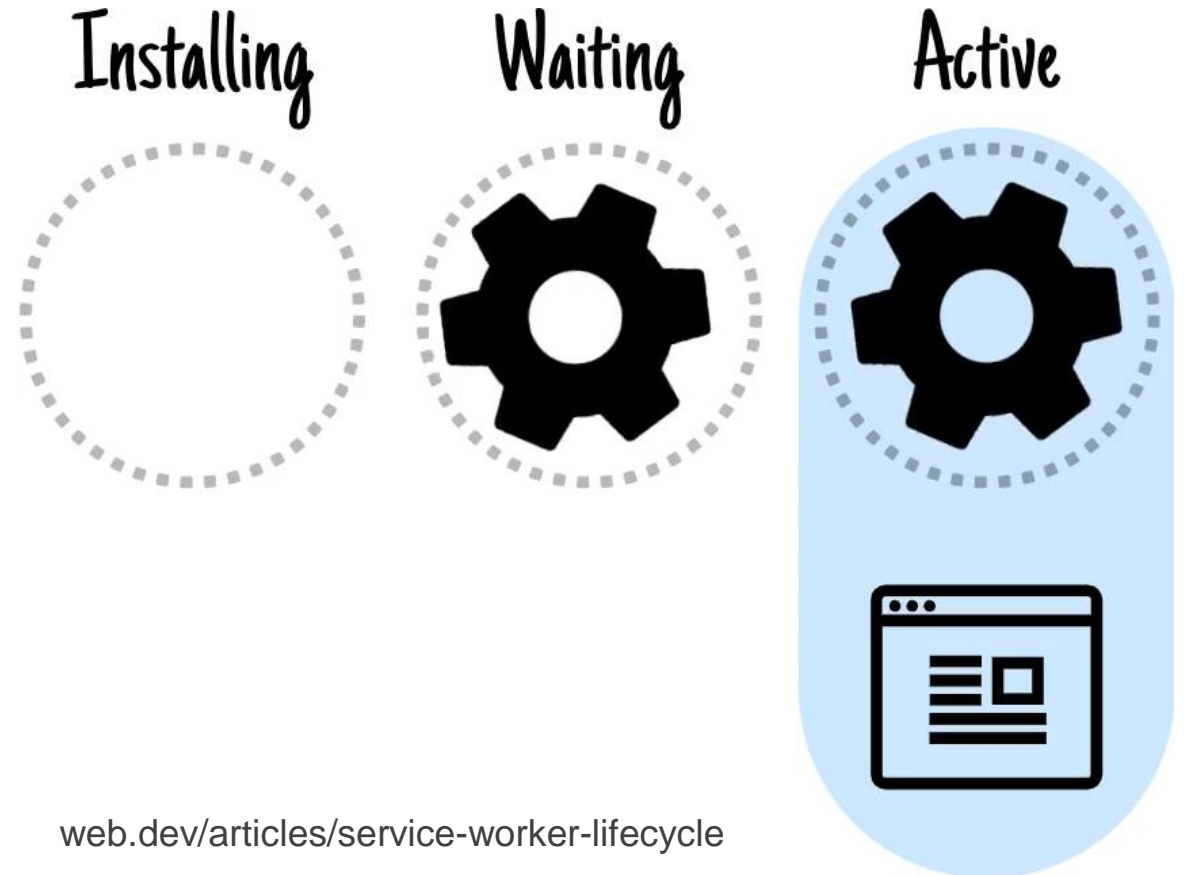
Active



web.dev/articles/service-worker-lifecycle

SERVICE WORKER UPDATE

- Neuer „Waiting“ Zustand
 - Der neue Service-Worker wartet auf einen App-Neustart um die Kontrolle zu übernehmen
- Update anstoßen durch einfaches Ändern des Codes der als Service Worker registrierten Datei
 - URL dabei nicht ändern!



SERVICE WORKER CODE

```
console.log("Bleep Bloop 🤖 -- hier spricht der ServiceWorker 🛠️");
```

👤 Oliver W

```
self.addEventListener({ type: "install", listener: event : Event => {  
    console.log("Der ServiceWorker wird installiert 🛠️");  
}});
```

👤 Oliver W

```
self.addEventListener({ type: "activate", listener: event : Event => {  
    console.log("Der ServiceWorker wird aktiviert 🚀");  
}});
```

AUFGABE 4 - DER ERSTE SERVICEWORKER

Wir registrieren einen einfachen Service Worker für unsere PWA.

1. Schreibt einen einfachen Service Worker welcher eine Nachricht in die Konsole schreibt wenn er installiert und aktiviert wird.
 2. Registriert diesen Service Worker in eurer App.
- Genauere Hinweise in der README.md im Ordner „Aufgabe 4“

Ihr habt hierfür ca. **10 Minuten**

SERVICE WORKER NICE TO KNOW

- Kein Zugriff auf DOM, `localStorage`, `window` / `document` etc.
- Service Worker sind „Event Driven“
 - Wichtige Events:
 - **install** / **activate**: Fürs Lifecycle management
 - **fetch**: Wenn die Anwendung eine Netzwerkressource laden will. Kern der Caching Funktionalität
 - **push**: Für Nachrichten aus dem Hintergrund, z.B. Push Benachrichtigungen via Web-Push
 - „**waitUntil()**“ und Promises benutzen
- Kein dynamisches **import()**, nur statische **import** Statements
- HTTPS ist eine Voraussetzung

CACHING MIT DEM SERVICE WORKER

- Keine Begrenzung der Cache Anzahl
- Identifiziert über einen Namen
- Inhalt ist eine „Map“ aus Request und Response Objekt Paaren
- Wichtig: sinnvolle Cache Versionierung bei Service Worker Updates!
 - Es sollte immer nur ein Service Worker gleichzeitig „in“ einem Cache sein
 - Kritische Phasen: Update mit manuellem Überspringen des Wartens (**skipWaiting()**)
 - Aufräumen alter Caches in der **activate** Phase
 - Speicherplatz ist auch nicht unendlich!

CACHING CODE - VORBEREITUNG

```
console.info(data: 'Bleep Bloop 🤖 - hier spricht der ServiceWorker 🛠️')

const cacheVersion : number = 1
const staticCache : string = 'static-cache-v' + cacheVersion

const staticFilesToCache : string[] = [
  './',
  './index.html',
  './style.css',
]
```

CACHING CODE - CACHES VORBEREITEN IM INSTALL

```
self.addEventListener('install', event : Event => {  
  console.info('Der ServiceWorker wird installiert. 🛠️')  
  
  event.waitUntil(  
    f: caches.open(cacheName: staticCache).then(cache : Cache => {  
      return cache.addAll(requests: staticFilesToCache)  
    })  
  )  
})
```

CACHING CODE - CACHES AUFRÄUMEN IM ACTIVATE

```
self.addEventListener('activate', event => {  
  console.info('Der ServiceWorker wird aktiviert. 🚀')  
  
  event.waitUntil(  
    f: caches.keys().then(cacheNames => {  
      return Promise.all(  
        values: cacheNames.map(cacheName => {  
          if (cacheName !== 'staticCache')  
            return caches.delete(cacheName)  
        })  
      })  
    })  
  })  
})
```


CACHING CODE - FETCH

```
self.addEventListener('fetch', event => {
  event.respondWith(
    r: tryToLoadFromNetwork(request: event.request, timeout: 10000)
      .catch(() => {
        console.info('Fehler beim Laden aus dem Netzwerk, versuche aus dem Cache zu laden...')
        .....
        return caches.open(cacheName: staticCache)
          .then(cache => {
            return cache.match(request: event.request, options: {ignoreSearch: true})
              .then(response: Response | undefined => {
                return response || new Response({body: null, init: {status: 404, statusText: 'Not found'}})
              })
          })
      })
  )
})
})
```

CACHING CODE - tryToLoadFromNetwork

```
function tryToLoadFromNetwork(request, timeout : number = 10000) : Promise<Response> {  
  console.info( data: `Versuche Datei ${request.url} aus dem Netzwerk zu laden... 🌐` )  
  return new Promise( executor: (resolve, reject) : void => {  
    const timeoutId : number = setTimeout( handler: reject, timeout: timeout )  
    fetch( input: request ).then( response : Response => {  
      clearTimeout( id: timeoutId )  
  
      if ( staticFilesToCache.includes( request.url ) ) {  
        updateStaticCache( request: request, response: response.clone() )  
      }  
  
      resolve( value: response )  
    } ).catch( reject )  
  } )  
}
```

CACHING CODE - updateStaticCache

```
function updateStaticCache(request, response) : void { Show usage
  caches.open({ cacheName: staticCache })
    .then(cache : Cache => {
      return cache.put({ request: request, response: response })
    })
}
```

AUFGABE 5 - CACHING!

Wir implementieren einen Service Worker mit einer „Network First“ Cache Strategie für die wichtigsten Dateien unserer App.

1. Erweitert euren Service Worker sodass er **index.html** und **style.css** in einem Cache nach der „Network First“ Strategie speichert.
2. Probiert das Ganze aus, z.B. mit dem Flugmodus eures Smartphones
 - Tipps und Hinweise in der README.md im Ordner „Aufgabe 5“

Ihr habt hierfür ca. **20 Minuten**

DAS RABBIT-HOLE IST SEHR TIEF!

- Bluetooth, Audio, Speicher, Hintergrund Sync, Biometrics, NFC, Geolocation, Push-Benachrichtigungen etc.: whatpwacando.today
 - Vieles davon ohne Service Worker
 - Web-Push, Offline Betrieb und Background Sync z.B. brauchen ihn aber
 - Tipp: „[Workbox](https://workbox.dev/)“ von Google als Library für Service Worker
- findpwa.com: Eine Sammlung von PWAs
- [PWAs in App-Stores](#) via [PWABuilder](#)



ICH HOFFE IHR HATTET SPAß UND HABT WAS GELEHRT!

PWA



Web App
Manifest

