



PWA – Progressive Web App

2 TAGES WORKSHOP



Über mich

Florian Hahn, 26 Jahre, Chemnitz

B. Sc. Medieninformatik

M. Sc. Webengineering

Wissenschaftlicher Mitarbeiter TU Chemnitz –
Professur Datenverwaltungssysteme

Freiberuflicher Dozent und Entwickler für Web

Promotion, Kunden und Praktika

Über Sie

Erwartungen / Erfahrungen

Prozess

Was werden wir lernen?

PWA Definitionen

Service Worker

Browser Tool

Events

Current State / Zukunft

Gliederung

1. Was ist PWA?
2. Einsatz
3. Vorteile
4. Nachteile
5. State of the Art
6. Features
7. Weiterführungen
8. Registrierung
9. Testing
10. Ausblick

Gliederung

Inhalt:

14 Stunden unterteilt in:

Grober Ablauf:

8 Stunden Theorie (10 Einheiten)

2 Stunden Übungen (3 Übungen)

2 Stunden Live Coding (5 Projekte)

2 Stunden Pausen (15+30+15 pro Tag)

Gliederung

1. Was ist PWA?

Einführung PWA

2015 wurden die Technologien von PWA's auf einer Konferenz von Google vorgestellt

Viele Tech Unternehmen investierten direkt in diese Technologie

Gartner prognostizierte: „...dass 50 % aller nativen Apps für Endanwender bis 2020 durch PWAs ersetzt werden.“

Dies ist bekanntlich nicht eingetreten

Was ist PWA?

PWA steht für Progressive Web App, seit 2016

Gegensatz zu Native App für Android und iOS

Läuft auf verschiedenen Betriebssystemen

Funktionen wie Offline Mode, Push Notification, etc.

Zugang einer normalen Web Application vom Home-Screen / Desktop

Zugang zu normalen Web Applications

Hat 4 spezifische Merkmale

Was ist PWA?

„Progressive“ steht für die Erweiterung einer Web Application und für den Zugang aus jedem Browser (alter Browser Support Einschränkung)

Entwicklungsmerkmale werden eindeutig und grundlegend definiert

ServiceWorker ist zentraler Bestandteil

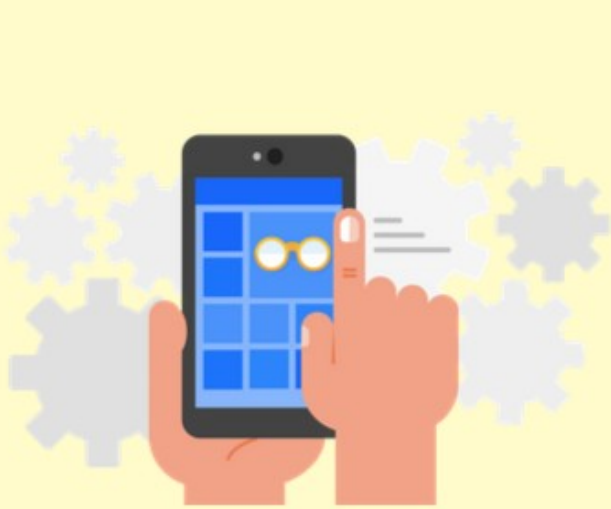
„Ein ServiceWorker beschreibt die Verarbeitung von Code des Browser im Hintergrund, um die Offline-Fähigkeit für Webanwendungen zu ermöglichen.“¹

Ein ServiceWorker übernimmt dabei durch Caching zwischen Request(Netzwerk) und Browser

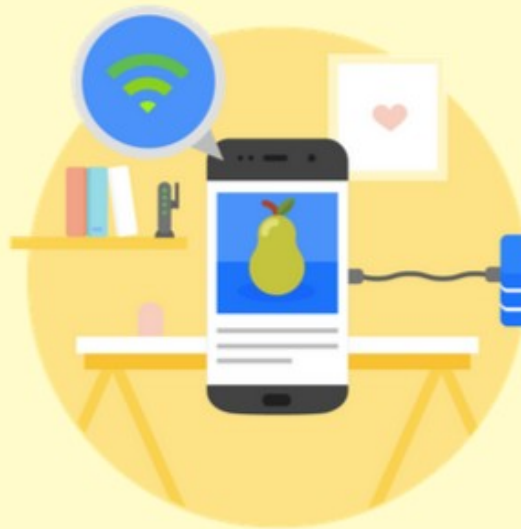
Der ServiceWorker SOLLTE im Browser Navigator enthalten sein

¹:Progressive Web Apps with React, Ausgabe Packt 2018, Seite 149

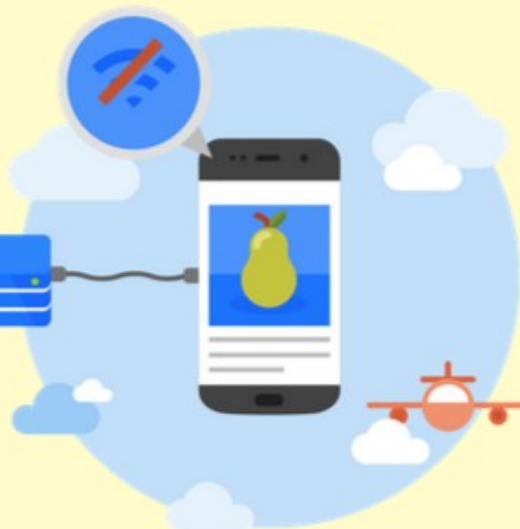
PROGRESSIVE WEB APP



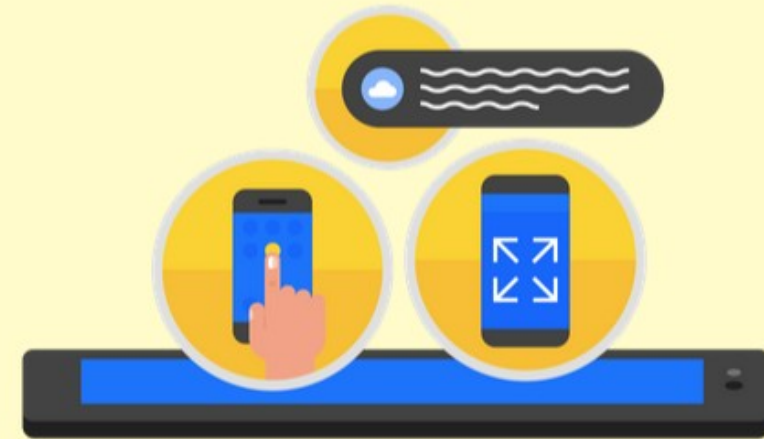
FAST



RELIABLE



INTEGRATED



ENGAGING

Gliederung

2. Einsatz

Einsatz

Schnelle und erreichbare Funktionen

Web Application Mobile Funktionen wie Cache, Push Notification, Offline, etc.

Home Zugang ohne Download

Globale Erreichbarkeit

Länder mit schlechter Internetanbindung

Schnelles Installation durch Endbenutzer

Browser Support für PWA bedeutet die Unterstützung von ServiceWorker über Navigator

Bedienbarkeit

Layout gleicht von Desktop zu mobile Endgeräte mit jeweiligen Browsern

Companies that adopted PWA



Uber



Starbucks



Google
Developers



Pinterest



Twitter



The Financial
Times



The Washington
Post



Forbes

	Native App	Responsive Website	Progressive Web App
Functions Offline	✓	✗	✓
Push Notifications	✓	✗	✓
Installable on home screen	✓	✗	✓
Full screen experience	✓	✗	✓
Indexable by search engines	✗	✓	✓
One place to enter content	✗	✓	✓
Works across all devices	✗	✓	✓
No download required	✗	✓	✓
Doesn't require updates	✗	✓	✓

Beispiele

<https://progressivewebapproom.com/>

<https://react-pwa-workbox-example.vercel.app/>

<https://www.smashingmagazine.com/>

<https://app.starbucks.com/>

<https://app.ft.com/>

Gliederung

3. Vorteile

Vorteile

Allgemein

- überall installierbar – Desktop, iOS, Android
- Möglichkeit für Android Store oder Apple Store oder zu umgehen
- Home Zugang
- Sicherheit durch HTTPS Only
- Updates

Entwicklung

- eine Codebase für iOS, Android, Windows, Linux, macOS
- Service Worker für Events
- Manifest.json für Installation
- Einfache Erweiterung



Gliederung

4. Nachteile

Nachteile

Allgemein:

- Native Apps sind 2022 nicht mehr zu groß
- Wachsender Mobile Daten Markt
- Offene WLAN Zugänge erleichtern Downloads und Datenmengen

Entwicklung:

- Spezifikationen für Manifest.json und ServiceWorker
- Registrierung (TWA = Trusted Web App)
- neues Tool Setup
- Navigator Check
- nur HTTPS Zugang

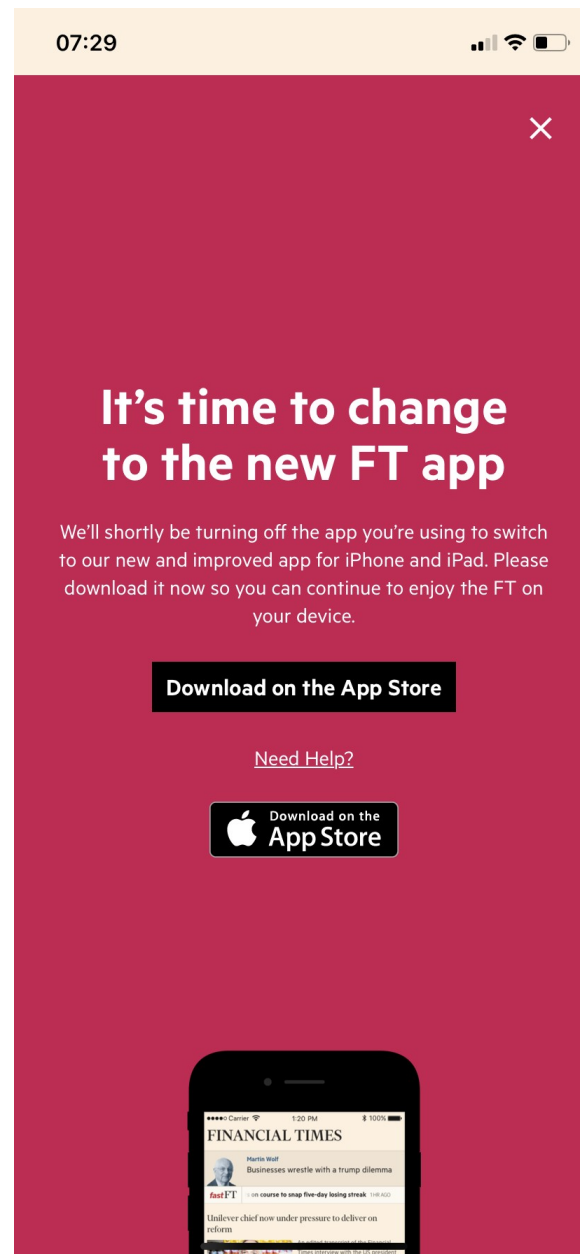
Nachteile

Allgemein:

- Benutzer Akzeptanz
- Browser Unterstützung
- Apple's Strategie
- Keine Store Revision – kann auch Vorteil sein

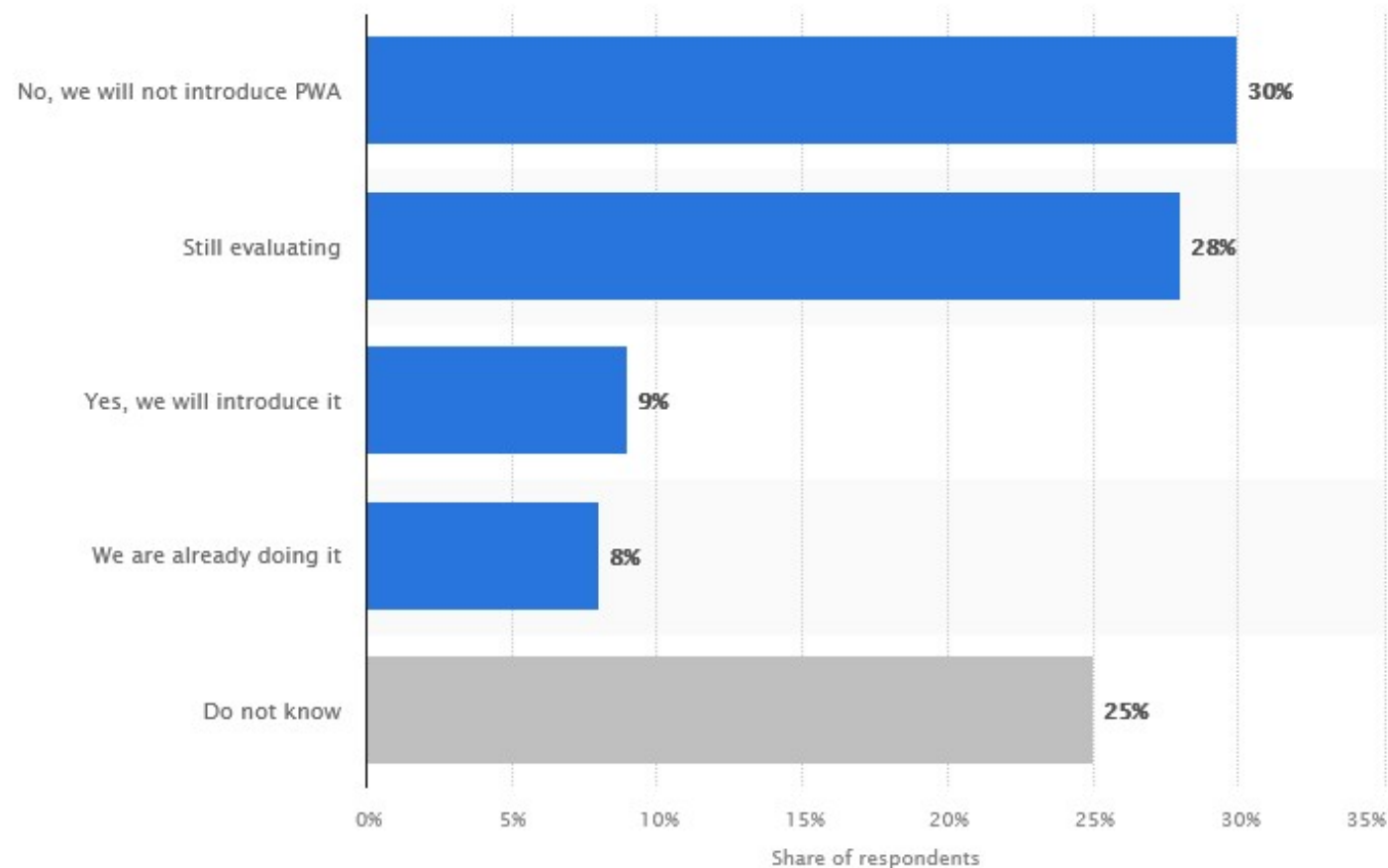
Entwicklung:

- Service Worker Lifecycle
- Browser Firmen kümmern sich kaum



eigener Screenshot von FT PWA, 18.08.2022

Umfrage in Tech-Firmen in Statista von Juli 2021, ob diese PWA nutzen/nutzen werden



[Additional Information](#)

© Statista 2022

[Show source](#)

Gartner, “Key Considerations When Building Web, Native or Hybrid Mobile Apps

Prognose:

„... dass bis zum Jahr 2025 90 % der Unternehmen eine Kombination aus Web-, nativen und hybriden Architekturen verwenden werden.“¹

Grundgedanke von Nativ versus Web versus PWA versus Hybrid

Zusammenfassung:

Keine genaue Zielfestlegung, PWA geht etwas unter gegenüber zukünftigen Nativen Ansätzen und Lösungen oder sogar gegen Hybride

Meinung: Individuelle hybride Lösungen basierend auf der Architektur

¹ Gartner, “Key Considerations When Building Web, Native or Hybrid Mobile Apps. Adrian Leow, Jason Wong. August 5, 2021.

Gliederung

5. State of the Art

State of the Art

Merkmale:

Generelle PWA Merkmale:

- Läuft über HTTPS (oder non-HTTPS localhost)
- Besitzt einen Service Worker
- Besitzt eine manifest.json
- Responsive und App ähnliche Darstellung

Gute PWA Merkmale:

- Offline/Caching
- Zugriff auf Hardware/Files/Sensoren
- Push-Benachrichtigung
- Installation/Hinzufügen

State of the Art

Minimales Projekt Beispiel für generelle PWA

Service Worker

Workbox

Web Manifest

Live-Demonstration

Was ist Workbox?

Workbox ist eine Library von Google Developers für PWA Service Worker

Aufgaben dieser Service Worker können Routing und Caching sein – Cache Strategien

Kann über Packet Manager installiert werden

Workbox bietet auch eine CLI an

Bereitstellung von injectManifest für eigene Service Worker

Was ist ein Service Worker?

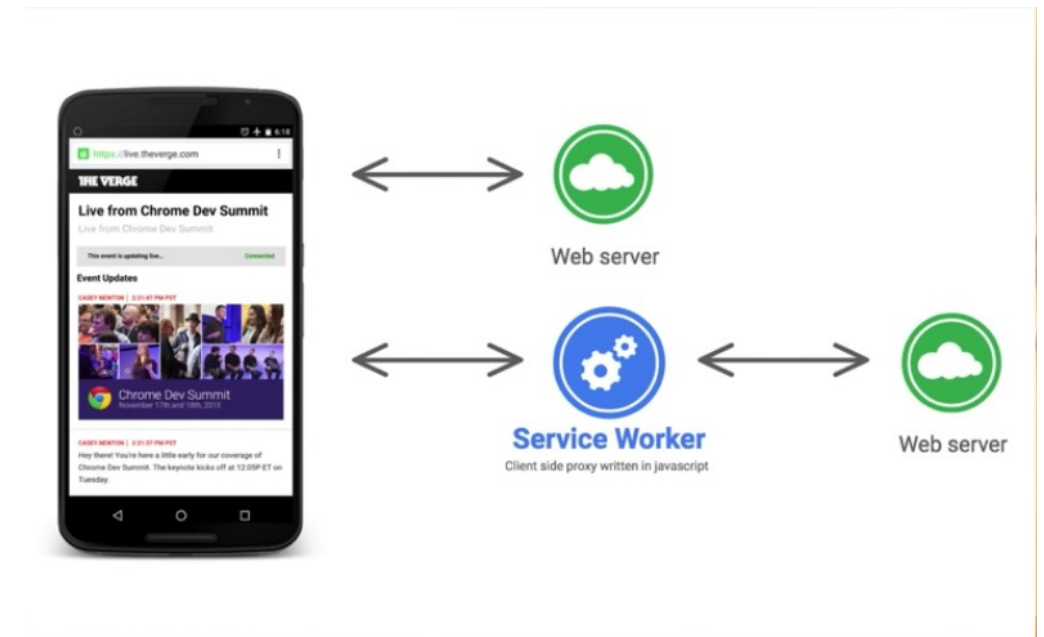
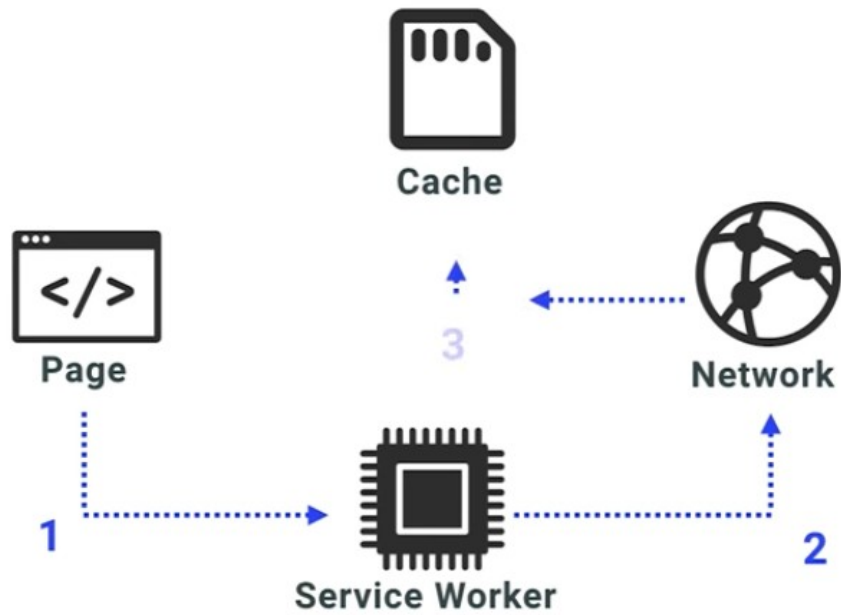
Ein Service Worker beschreibt und verarbeitet die Event Händler

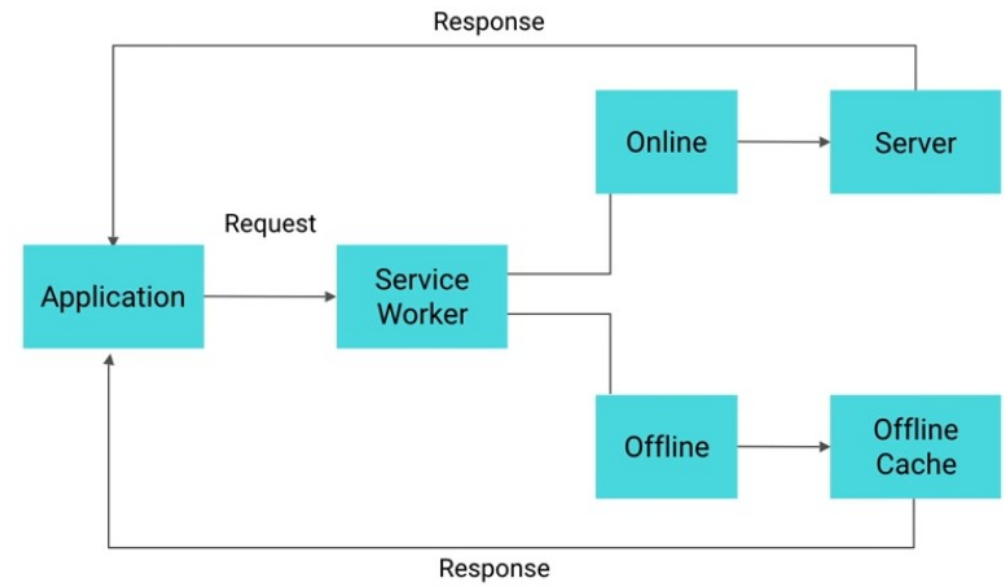
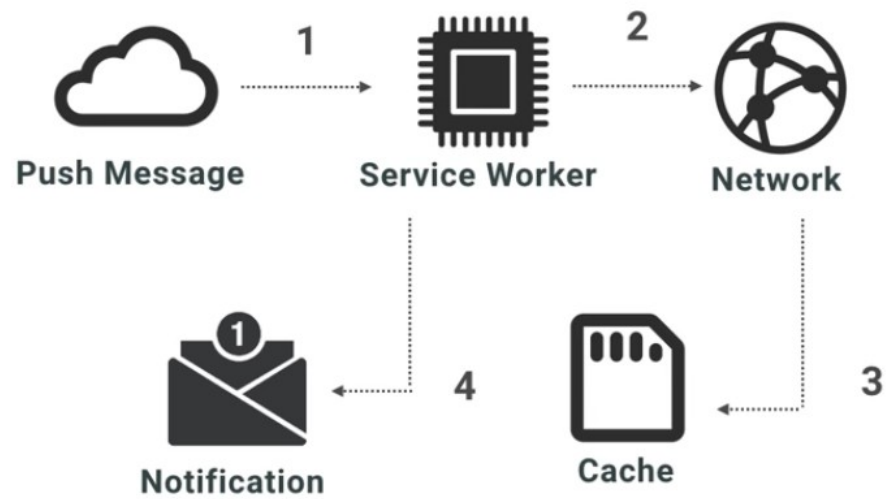
Er bildet eine Schnittstelle zwischen Browser und Netzwerk

Lifecycle

Events:

- Fetch – Netzwerk Eingriffe für Cache
- Push – PushAPI und NotificationAPI
- Sync – SyncManager für Scheduling





Was ist ein Service Worker Lifecycle?

Ein Service Worker LifeCycle beschreibt den Vorgang für eine Lifecycle

Er definiert die Event Abfolge

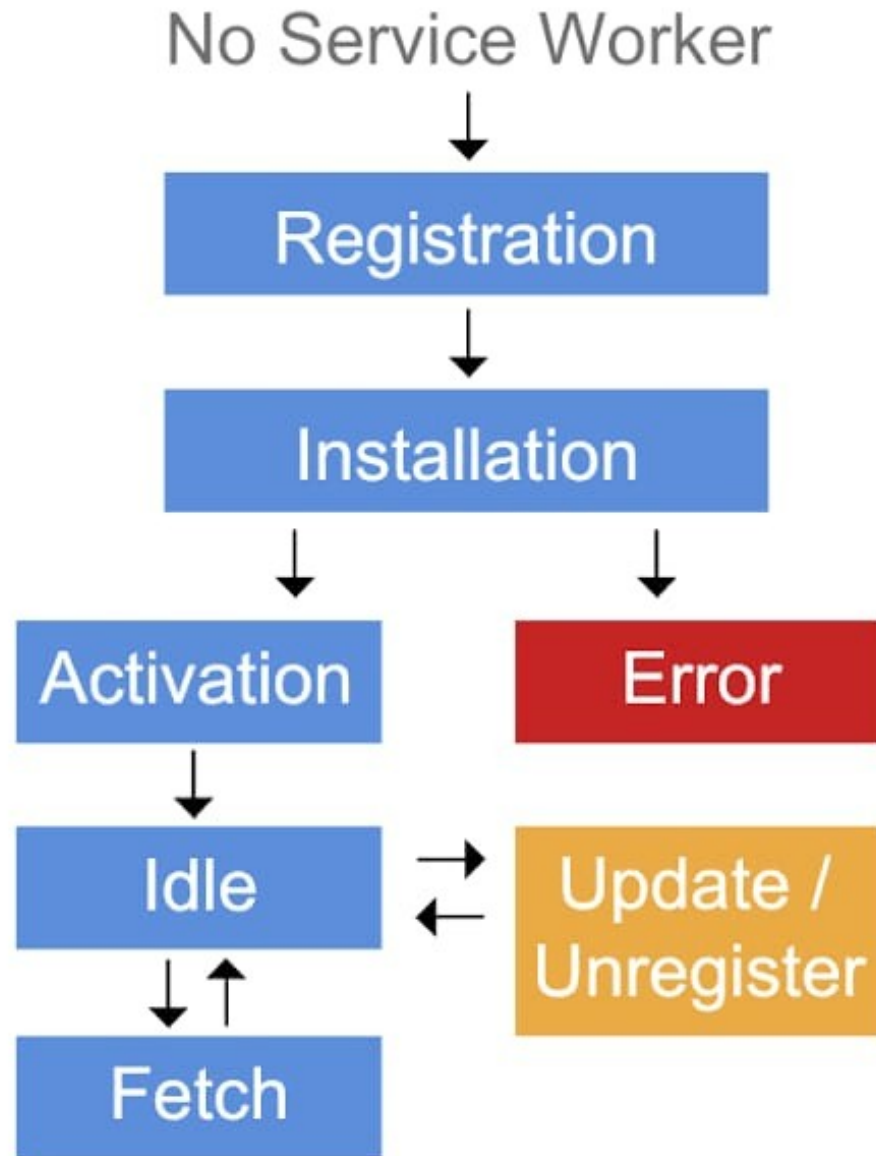
Es ist immer ein Service Worker in einen extra CPU Thread geöffnet

Hat keinen Zugriff auf das DOM

Immer nur EIN Service Worker aktiv

Wird in einer Registrierung auf die Service Worker Datei verwiesen

Hinweis: `importScripts()`



Service Worker Registrierung

Navigator Check

```
if (!('serviceWorker' in navigator)) {  
  console.log('sw not supported');  
  return;  
}
```

main.js

Datei Registrierung

```
navigator.serviceWorker.register(  
  '/service-worker.js'  
)
```

Service Worker Scope

```
.then(function(registration) {  
  console.log('SW registered! Scope is:',  
    registration.scope);  
});
```

Ist Root

Kann auch Options Scope
sein

```
});  
// .catch a registration error
```

Service Worker Registrierung

Hinweis zur Registrierung:

Registrierung kann zu jeder Zeit durchgeführt werden

DOM Events, Manuell, Time Events, onUpdate

Best Practice: DOM Event „load“

Grund: Für den Nebenthread werden trotzdem Ressourcen verbraucht und für Caching muss zuerst die richtige Datei im Browser geladen und angezeigt sein

Nach der Registrierung wird „self“ im Service Worker als „window“ betrachtet

Service Worker Installation

Self verweist auf „window“

Events sind hier aufgrund der Service Worker API's verfügbar

Beginnt nach der Registrierung im Service Worker

Zeitpunkt für z.B. Caching

```
self.addEventListener('install',  
function(event) {  
    // Do stuff during install  
});
```

service-worker.js

Service Worker Installation

Hinweis zur Installation:

Installation wird immer direkt als erstes im LifeCycle nach der Registrierung getriggert

Verschiedene Service Worker API's: install, activate, push,

Best Practice: install und activate ganz oben im Service Worker

Innerhalb der Installation können wir mit dem „event“ auf verschiedene Methoden zugreifen

Event.waitUntil wartet bis der Code ausgeführt ist und installiert dann den Service Worker

localStorage/sessionStorage/Cookies kann nicht genutzt werden, da kein synchroner Code hier erlaubt ist

Cache und IndexedDB ist hingegen erlaubt

Service Worker Installation

Cache Zugriff über „caches“

Definition wie bei Storage über Namen, z.B. „v1“

Methoden wie `event.waitUntil()` und `event.respondWith()` geben IMMER ein Promise zurück

Problem: Es darf immer nur EIN Service Worker aktiv sein

Lösung: `skipWaiting()` überbrückt den Unterschied zwischen verschiedenen Service Worker Versionen durch Bit Unterschiede. Somit wird der Browser gezwungen, die neue Version zu benutzen. Dies ist oft ein großes Problem in der Entwicklung

Service Worker Aktivierung

Self verweist auf „window“

Events sind hier aufgrund der
Service Worker API's verfügbar

Aktivierung des ServiceWorker

Zeitpunkt für z.B. Löschen

```
self.addEventListener('activate',  
function(event) {  
    // Do stuff during install  
});
```

service-worker.js

Service Worker Aktivierung

Hinweis zur Aktivierung:

Die Aktivierung ist der letzte Schritt im Life Cycle

Durch die Aktivierung kann der Idle Zustand hergestellt werden

Innerhalb der Installation können wir mit dem „event“ auf verschiedene Methoden zugreifen

`Event.waitUntil` wartet bis der Code ausgeführt ist und aktiviert dann den Service Worker

Service Worker Events

Am meisten genutzte Events:

Install

Active

Fetch

Push

Sync

Weitere Beispiel:

Message

Web App Manifest

Ein wichtiger Bestandteil einer PWA

Einer der vier Merkmale einer PWA

Meistens manifest.json ODER manifest.webmanifest

In dieser Notation werden verschiedene Grundeinstellungen für die PWA geliefert

Sollte immer, wie der Service Worker, im Root liegen

Testing durch den Dev Tools

Unterstützung der „Application“ > „Manifest“ Darstellung über Service Worker

Alle Eigenschaften sind freiwillig, aber einige zu empfehlen

Einbindung über LINK HTML Tag Element

`<link rel=„manifest“ href=„manifest.json“>`

Web App Manifest

Eigenschaften:

*Name: Name der ausführbaren PWA auf Desktop oder App auf mobil

*Icons: Array aus vielen Icons der PWA, verschiedenen Auflösungen, verschiedene Browser

*Start_url: Legt die Landingpage der PWA fest

*Display: Wie das OS die PWA starten soll – fullscreen, standalone, minimal-ui

*ID: Definiert eine eindeutige ID der PWA gegenüber anderen PWA auf der Domain

Description: Beschreibt den Zweck der PWA

Screenshots: Array für aussagekräftige Bilder der PWA – oft Auflösungsprobleme

Theme_color: Standard Farbe der PWA z.B. oberer Bildschirmrand

Background_color: Farbe beim Splash Screen vom Ladevorgang und generelle Hintergrundfarbe

Short_name: Kürzerer Name als Name, falls Umbruch erfolgen muss

Web App Manifest

Icons:

Zentraler Bestandteil eines Manifest

Probleme für verschiedene Formate, Größen, Auflösungen und OS

Formate: in modernen Web Umfeld sollten folgende unterstützt werden: SVG und PNG

Auflösungen: verschiedene Auflösungen z.B. 192px X 192px und 512px X 512px

Größen: verschiedene Größen für Optimierung

Problem 1: Apple braucht sehr viele Icons für verschiedene Darstellungen

Problem 2: Apple akzeptiert KEINE manifest Icons, sondern NUR HTML Tag Elementen

z.B.: `<link rel="apple-touch-startup-image" href="icons/apple-splash-2048-2732.jpg" portrait="...">`

Persönliche Erfahrung: Wird höchstens einmal per Hand durchgeführt, deswegen...

Web App Manifest

Generatoren:

Packete:

Pwa-asset-generator

Automatische Erstellung von Icon Auflösungen und HTML Tag Elementen

Web:

<https://app-manifest.firebaseapp.com/>

Dynamische Eingabe der Werte und Upload der Icons

Live Coding Projekt 1

Live Coding Projekt 2

Übung 1

Aufgabe:

Erstellen Sie bitte eigenständig Ihren ersten eigenen Service Worker und schauen Sie sich den Life Cycle genau an!

Gerne können Sie die Ausgabe der einzelnen Zustände in einer Konsole ausgeben!

Hinweise:

Sie sind in der Auswahl der Frameworks/Sprachen nicht eingeschränkt!

Bitte verwenden Sie für Testing/Debugging Google Chrome!

Zeit: 30 Minuten

Gliederung

6. Features

Features

5.1 Offline Modus

5.2 Caching

5.3 Push Notification

5.4 Background Sync

5.5 File System

Offline Modus

Bekannte und meist genutzte Eigenschaft von PWA's

Kommt automatisch, sobald wir bestimmte Dateien zwischenspeichern

Wird auch von native Apps verwendet

Der Benutzer ist unabhängig von der Internet Verbindung

Funktionalität der Anwendung können entweder gar nicht oder nur eingeschränkt genutzt werden

Wird durch Caching erreicht

Offline Modus

Support von Offline Modus durch Caching:

	Android (Chrome)	Android (Firefox)	iOS (Safari)	Desktop (Chrome/Edge)	Desktop (Firefox)
Offline-Modus	Ja	Ja	Ja	Ja	Ja

Aufgrund der verschiedenen Möglichkeiten von Web Browser Speicherung, wird der Offline Modus durch Caching als einziges Feature von allen unterstützt.

Caching

Was ist ein Cache?

Ein Cache dient zur schnellen Zwischenspeicherung für den Browser oder Software

Inhalte können schneller angezeigt werden

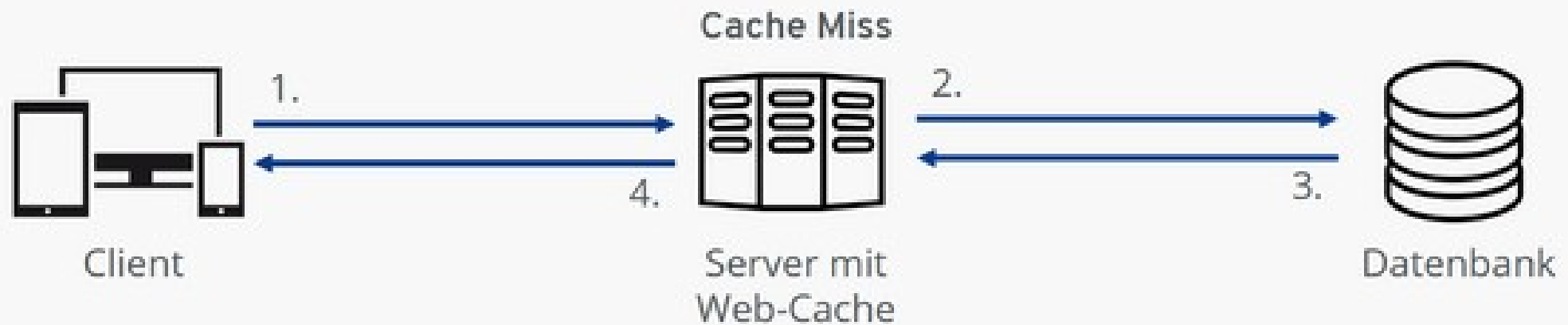
Erneutes herunterladen wird verhindert

Besonders bei mobilen Daten gefragt

Cache ist „versteckt“ (Entwickler Tools)

Es gibt CacheHit und CacheMiss

Es können pro Web Application mehrere Caches aktiv sein



Caching

Vorteile:

Offline Modus

Geschwindigkeit der Ladezeiten

Datenreduzierung

Nachteile:

Aktuelle Daten / Darstellungsfehler

Cache Invalidierung / Aktualisierung

CacheMiss

Caching

Arten von Caches:

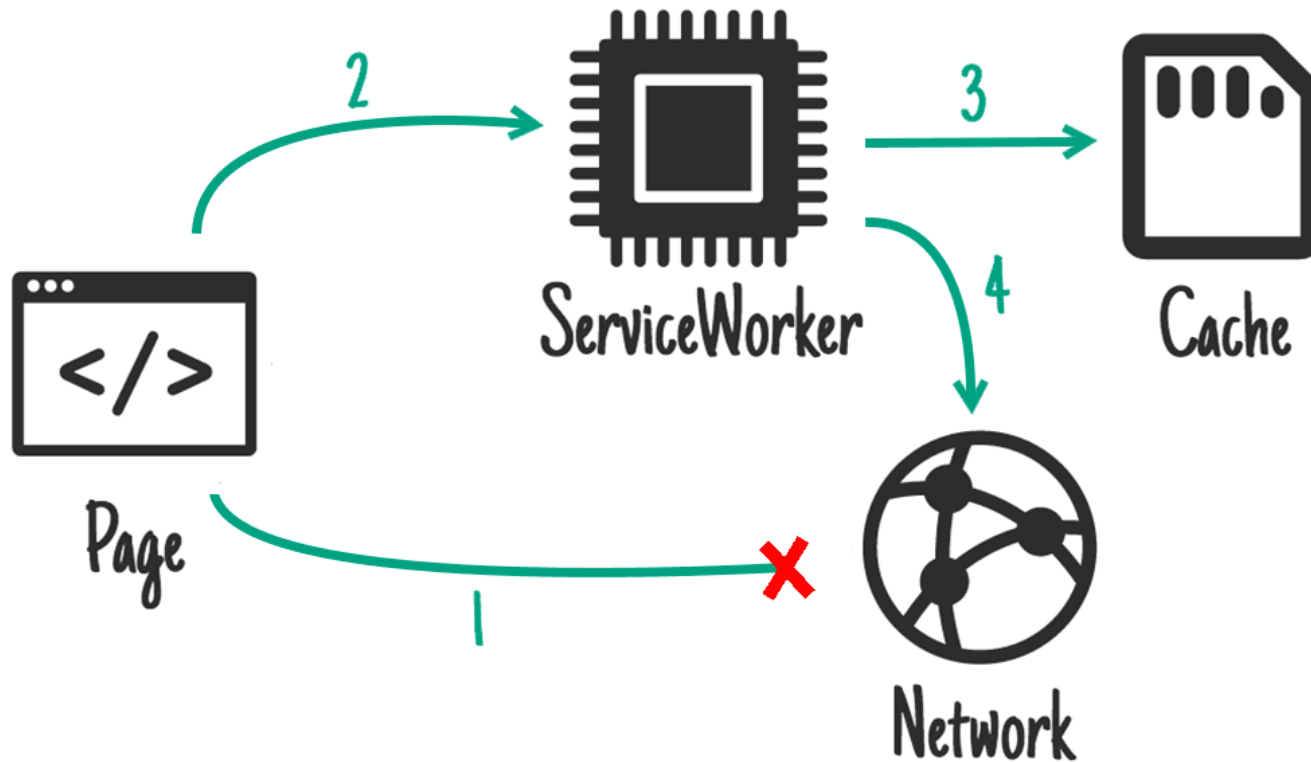
Ressource	Cache	Größe des Cache	Zugriffszeit mit Cache	× langsamer ohne Cache
Hauptspeicher	Level 1-Cache (Hardware)	Dutzende Kilobyte (KB)	Weniger als eine Nanosekunde (ns)	200 ×
Festplatte	Festplatten-Cache (Hardware)	Dutzende Megabyte (MB)	Hunderte Nanosekunden (ns)	100 ×
Browser	Browser-Cache (Software)	Mehrere Gigabyte (GB)	Dutzende Millisekunden (ms)	10–100 ×
Websites	CDNs, Google Page Cache, Wayback Machine (Software)	Tausende Terabyte (Petabyte, PB)	Wenige Sekunden (s)	2–5 ×

Browser (Web) Cache hält HTML, CSS, JS, Bilder und Fonts

Ablage im Arbeitsspeicher aus der Browser Software

Caching

Warum wird der Offline Modus durch Caching erreicht?



Caching

Speicher Möglichkeiten:

SessionStorage/LocalStorage versus Cookies versus Cache versus IndexedDB

SessionStorage/LocalStorage ist limitiert und hat keinen asynchronen Zugriff

Cookies haben immer ein Ablaufdatum und sind limitiert

IndexedDB hat eine schlechte API Zugriff und ist eine Client NoSQL DB

Cache ist unlimitiert und der Zugriff ist einfach

Caching

	Chrome	Firefox	Safari	Safari	IE	IE
	40	34	6, 7	8	9	10, 11
Application Cache	up to quota	500MB, Unlimited	Unlimited?	Unlimited?		100MB?
FileSystem	up to quota					
IndexedDB	up to quota	50MB, Unlimited		up to quota?		10MB, 250MB (~999MB)
WebSQL	up to quota		5MB, 10MB, 50MB, 100MB, 500MB, 600MB, 700MB...	5MB, 10MB, 50MB, 100MB, 500MB, 600MB, 700MB...		
LocalStorage	10MB	10MB	5MB	5MB	10MB	10MB
SessionStorage	10MB		Unlimited	Unlimited	10MB	10MB

Caching

Caching Strategien:

Wie der Service Worker auf die verschiedenen Netzwerk Anfragen reagiert

NetworkFirst, CacheFirst, CacheOnly, NetworkOnly, CacheNetwork, IgnoreBoth Möglichkeiten

CacheOnly greift NUR auf den Cache zu und gibt Anfragen aus dem Cache zurück

NetworkOnly gibt NUR Daten aus den Anfragen weiter

CacheFirst versucht alle Cache Daten zu laden, danach die Anfragen

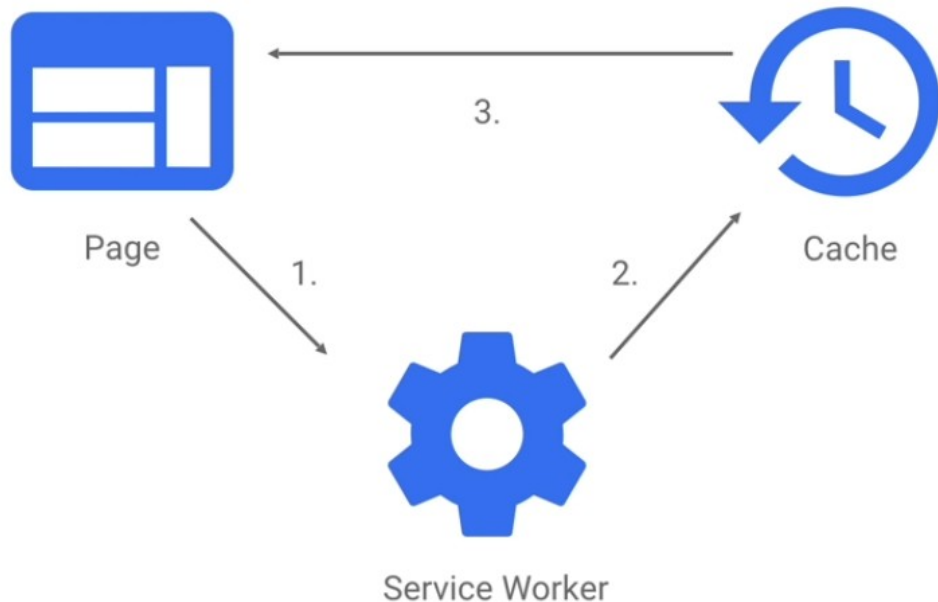
NetworkFirst versucht alle Daten zu laden und erst nach NetworkFail Zugriff auf Cache

CacheThenNetwork greift auf Cache und Anfragen zu und updated den Cache vor der Darstellung

Fallback reicht jede Anfrage durch und gibt eine Seite aus

Cache Only:

Cache Only



Cache Only - example code

```
self.addEventListener('fetch', event => {  
  // If a match isn't found in the cache, the response  
  // will look like a connection error  
  event.respondWith(caches.match(event.request));  
});
```

sw.js

Cache Only

Vorteile:

Schnelle Ladezeit

Gespeicherte Daten auf offline

Nachteile:

Keine Updates

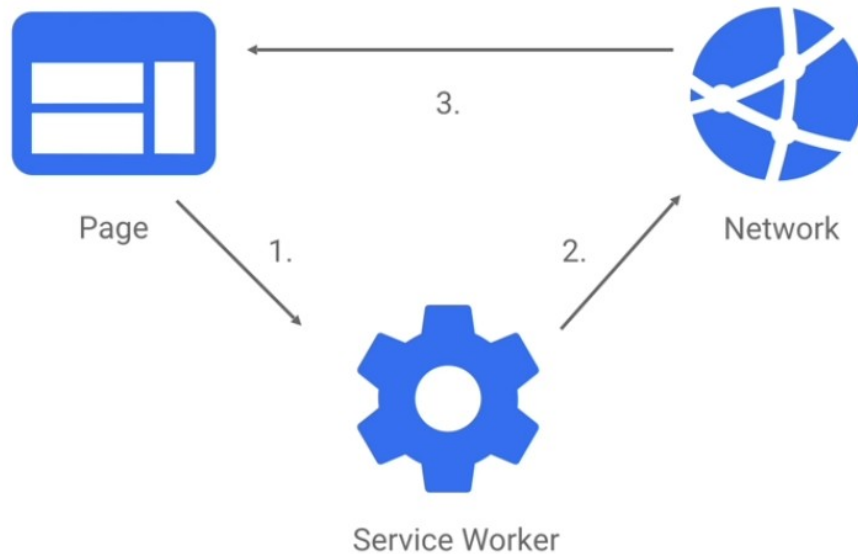
Keine neuen Daten

Eingeschränkte Funktionalität

Keine Interaktivität

Network Only:

Network Only



Network Only - example code

```
self.addEventListener('fetch', event => {  
  event.respondWith(fetch(event.request));  
  // or simply don't call event.respondWith, which  
  // will result in default browser behaviour  
});
```

sw.js

Network Only

Vorteile:

Volle Funktionalität

Jederzeit Updates

Voller Datenzugriff

Nachteile:

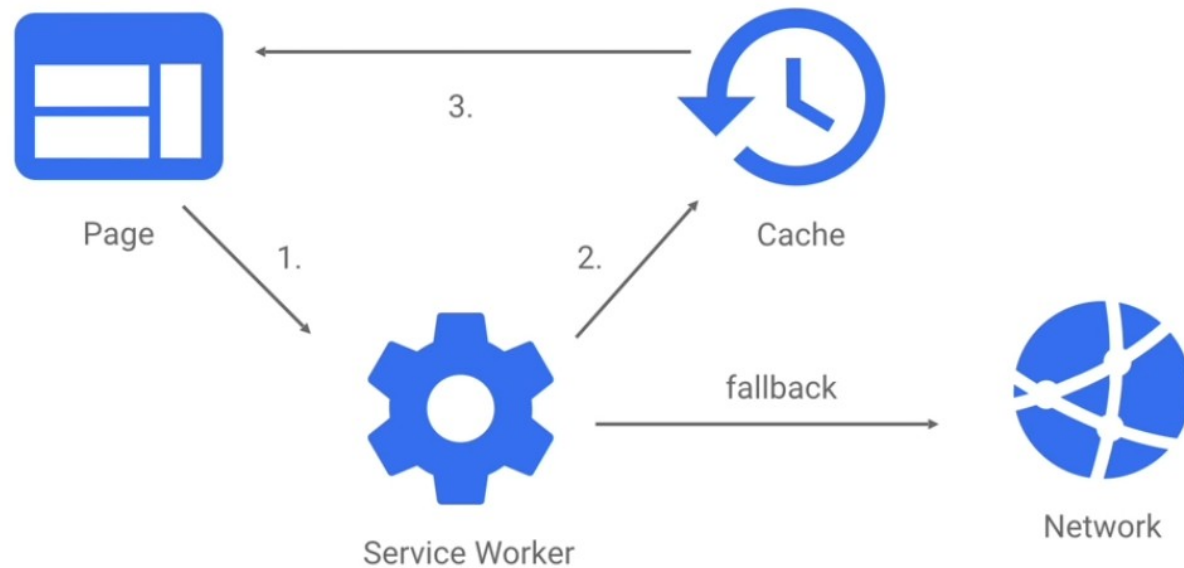
Langsame Ladezeit

Keine Cache Speicherung

Keine Offline Möglichkeit

Cache First

Cache First



Cache First - example code

```
self.addEventListener('fetch', event => {  
  event.respondWith(  
    caches.match(event.request).then(response => {  
      return response || fetch(event.request);  
    })  
  );  
});
```

sw.js

Cache First: - Ansatz von OfflineFirst !

Einsatz: (Static Webseiten, Unternehmen, One-Pager)

Vorteile:

Gute Ladezeiten

Offline Modus

Volle Funktionalität

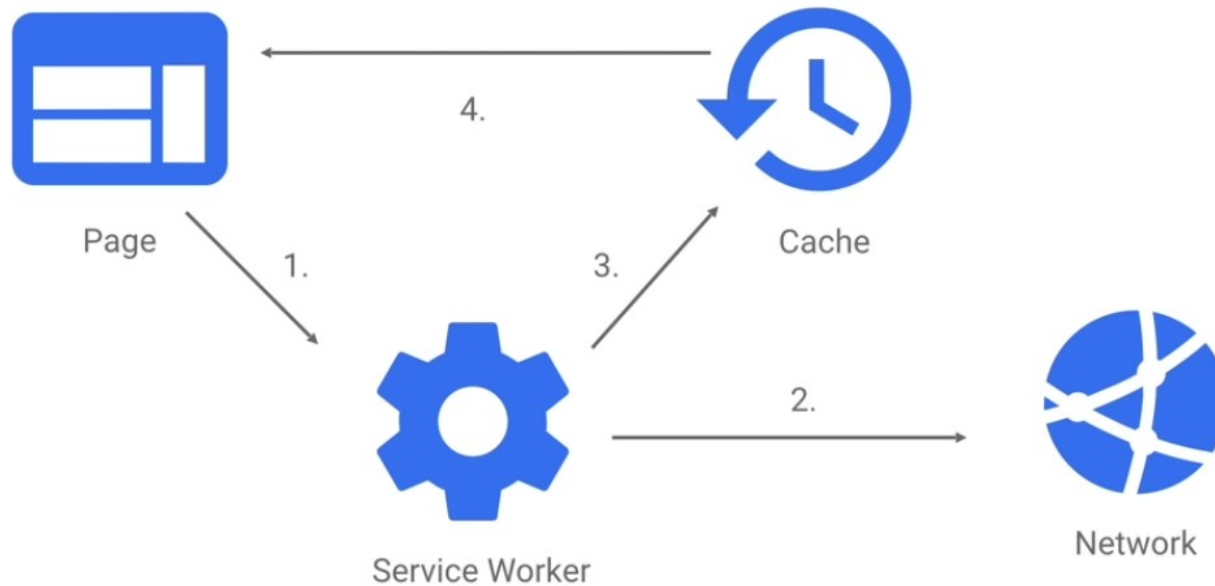
Nachteile:

Alte Daten werden dargestellt

Cache Updates

Network First:

Network First



Network First - example code

```
self.addEventListener('fetch', event => {  
  event.respondWith(  
    fetch(event.request).catch(() => {  
      return caches.match(event.request);  
    })  
  );  
});
```

sw.js

Network First

Einsatz: Chats, Social-Media, Shops

Vorteile:

Aktualität

Updates

Volle Funktionalität

Nachteile:

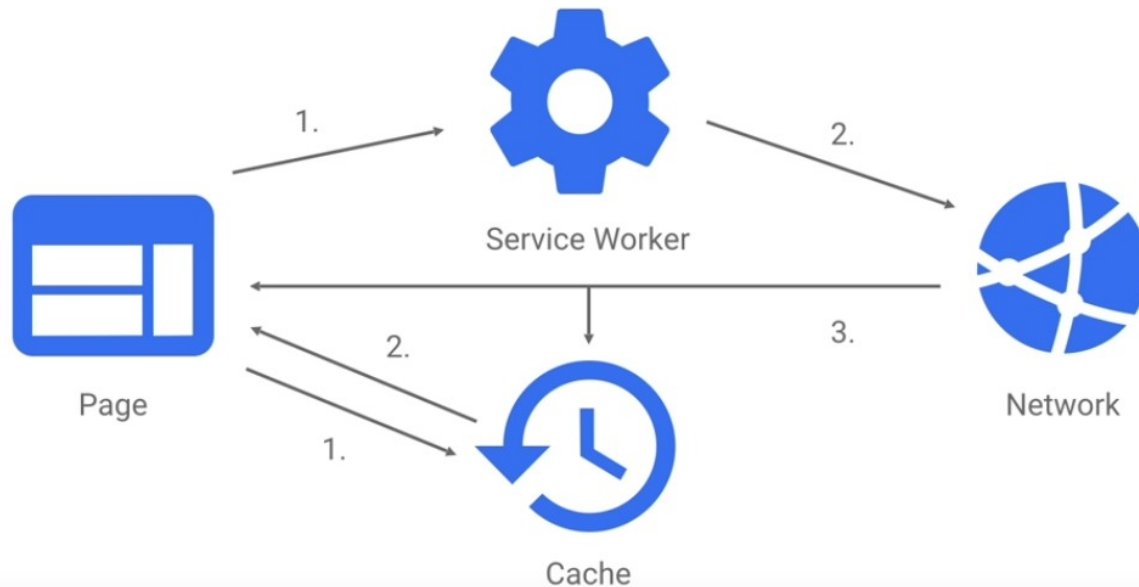
Langsame Ladezeit nach NetworkFailed

Cache Überschreibung

Eingeschränkter Offline Modus

Cache then Network:

Cache Then Network



Cache Then Network - example code

```
self.addEventListener('fetch', event => {  
  event.respondWith(  
    caches.open('mysite-dynamic').then(cache => {  
      return fetch(event.request).then(response => {  
        cache.put(event.request, response.clone());  
        return response;  
      });  
    })  
  );  
});
```

sw.js

Cache then Network

Einsatz: Spiele

Vorteile:

Aktualität und Update

Offline Modus

Volle Funktionalität

Nachteile:

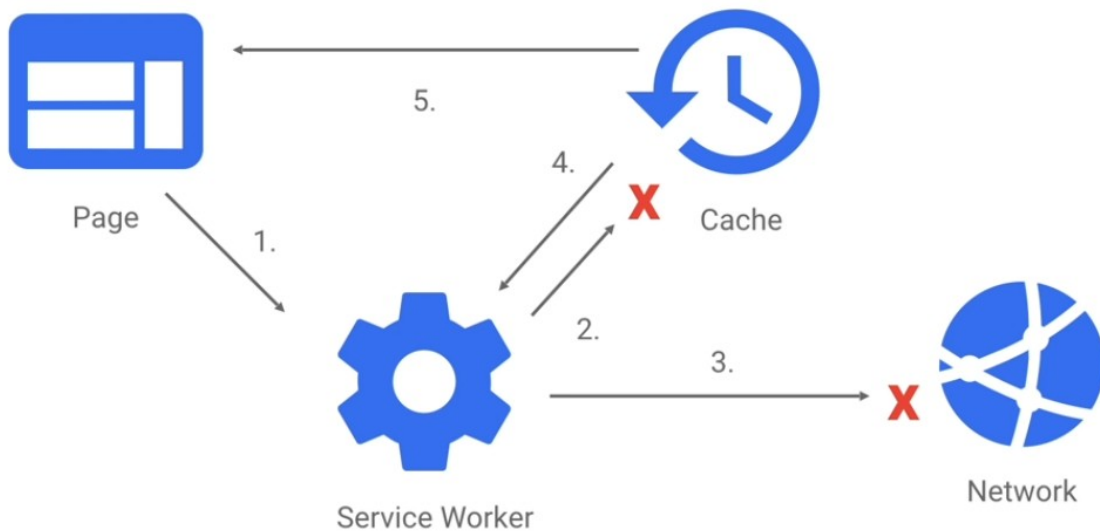
Lost Update

Zugriff auf Cache/IndexedDB als Frontend Datenspeicherung und Update – kein DBMS (ACID)

Große Datenmengen

Fallback:

Generic Fallback



Generic Fallback - example code

```
self.addEventListener('fetch', event => {  
  event.respondWith(  
    caches.match(event.request).then(response => {  
      return response || fetch(event.request);  
    }).catch(() => {  
      return caches.match('/offline.html');  
    })  
  );  
});
```

sw.js

Fallback

Vorteile:

Backup Lösung

Kann im Promise ge-catch-ed werden

Ist kombinierbar

Nachteile:

Erweiterung Logik

Workbox

Workbox als CDN für die Benutzung von einfachen Cache Strategien im Service Worker
Workbox unterstützt alle eben genannten Strategien

Using built in strategies

```
importScripts(  
  'https://storage.googleapis.com/.../workbox-sw.js'  
);  
workbox.routing(  
  new RegExp('/images/avatars/'),  
  workbox.strategies.staleWhileRevalidate()  
);
```

A blue rectangular button with the text "sw.js" in white, representing a file or a link to the service worker script.

Caching

Wann und wie den Cache updaten:

Die Cache API wird vom Browser angesteuert

Wir können den Cache bei Events (install, active, fetch) oder bei Interaktionen updaten

Jeder Cache besitzt einen Cache Namen, welcher oft versioniert ist

Zusätzlich können wir im Cache verschiedene Dateitypen oder Ordner speichern

Der Cache kann entweder eine Datei oder mehrere Dateien hinzugefügt werden

Jeder Cache Zugriff basiert auf Promises

Best Practice: Im install Life Cycle wird über `waitUntil` der Cache befüllt

Caching

Preaching mit Cache Namen und Dateien / Ordner:

```
const cacheName = 'cache-v1';  
const resourcesToPrecache = [  
  '/',  
  'index.html',  
  'styles/main.css',  
  'images/space1.jpg',  
  'images/space2.jpg',  
  'images/space3.jpg'  
];
```

A blue rectangular button with the text "sw.js" in white, representing a service worker file.

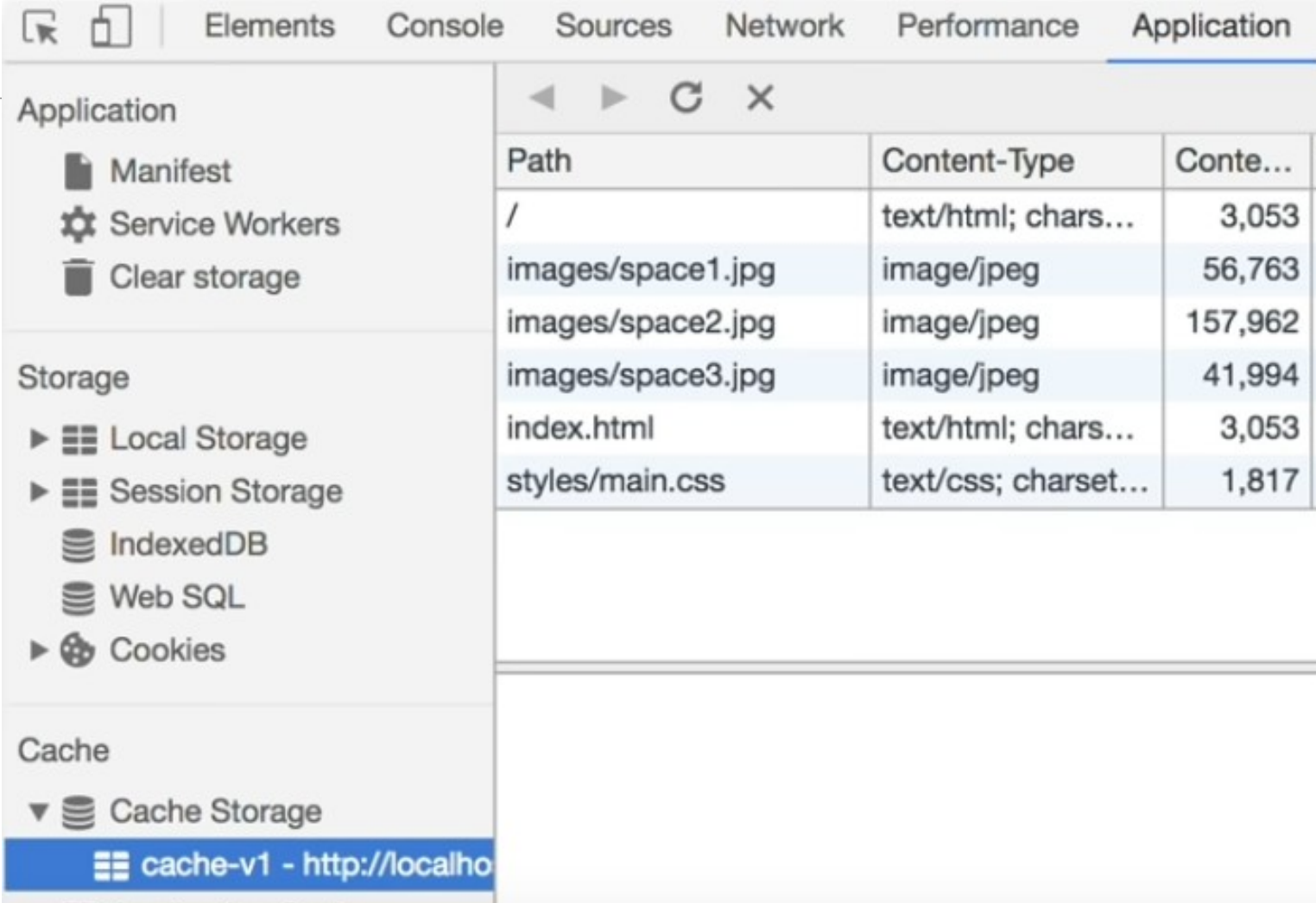
Caching

Cache Zugriff speichern bei install Event:

```
self.addEventListener('install', event => {  
  event.waitUntil(  
    caches.open(cacheName).then(cache => {  
      return cache.addAll(filesToCache);  
    })  
  );  
});
```

Caching

Cache im Browser (Dev Tools):



The screenshot shows the Chrome DevTools Application tab. The left sidebar has three sections: Application, Storage, and Cache. The Application section contains 'Manifest', 'Service Workers', and 'Clear storage'. The Storage section contains 'Local Storage', 'Session Storage', 'IndexedDB', 'Web SQL', and 'Cookies'. The Cache section contains 'Cache Storage' and 'cache-v1 - http://localhost'. The right pane shows a table of cached resources.

Path	Content-Type	Content Size
/	text/html; charset=...	3,053
images/space1.jpg	image/jpeg	56,763
images/space2.jpg	image/jpeg	157,962
images/space3.jpg	image/jpeg	41,994
index.html	text/html; charset=...	3,053
styles/main.css	text/css; charset=...	1,817

Caching

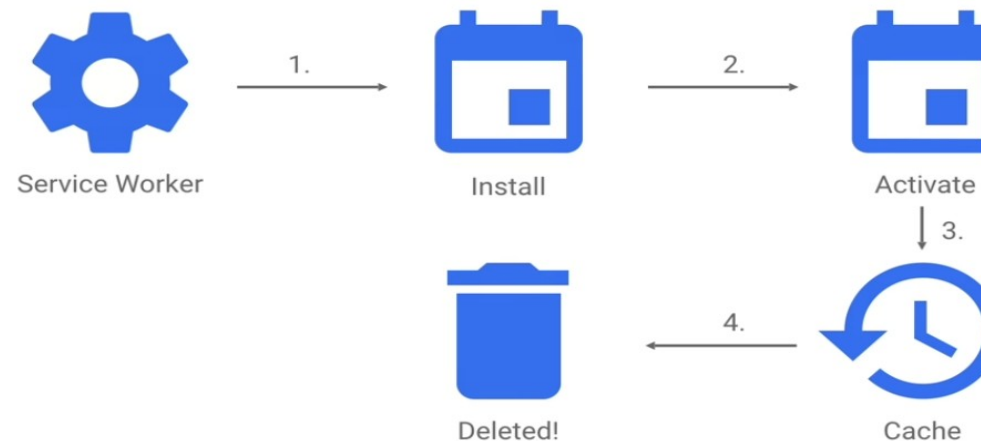
Cache Zugriff löschen:

Bei verschiedenen Caching Strategien möchten wir alte Caches löschen/erneuern

Bei dem Event „activate“, welches immer läuft können wir dies durchführen

Überprüfung einer zu erwartenden Version gegenüber der aktuellen Version

Diese Versionierung kann durch eine API mit einem DBMS unterstützt werden (online)



Cache Zugriff löschen

```
self.addEventListener('activate', event => {  
  // delete any caches that aren't in expectedCaches  
  // which will get rid of v1  
  event.waitUntil(  
    caches.keys().then(keys => Promise.all(  
      keys.map(key => {  
        if (!expectedCaches.includes(key)) {  
          return caches.delete(key);  
        }  
      })  
    )).then(() => {  
      console.log('SW now ready to handle fetches!');  
    })  
  );  
});
```


Caching

Möglichkeiten für andere Caching Optionen:

User Interaktion

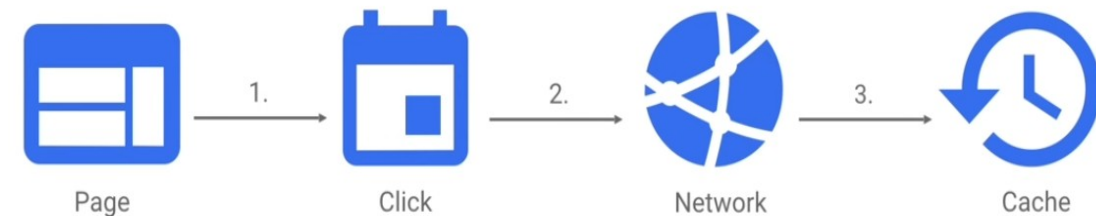
Online Status / Anfragen – Cache then Network

Caching Strategien – in einer gewisser Zeitangabe

Push Notification

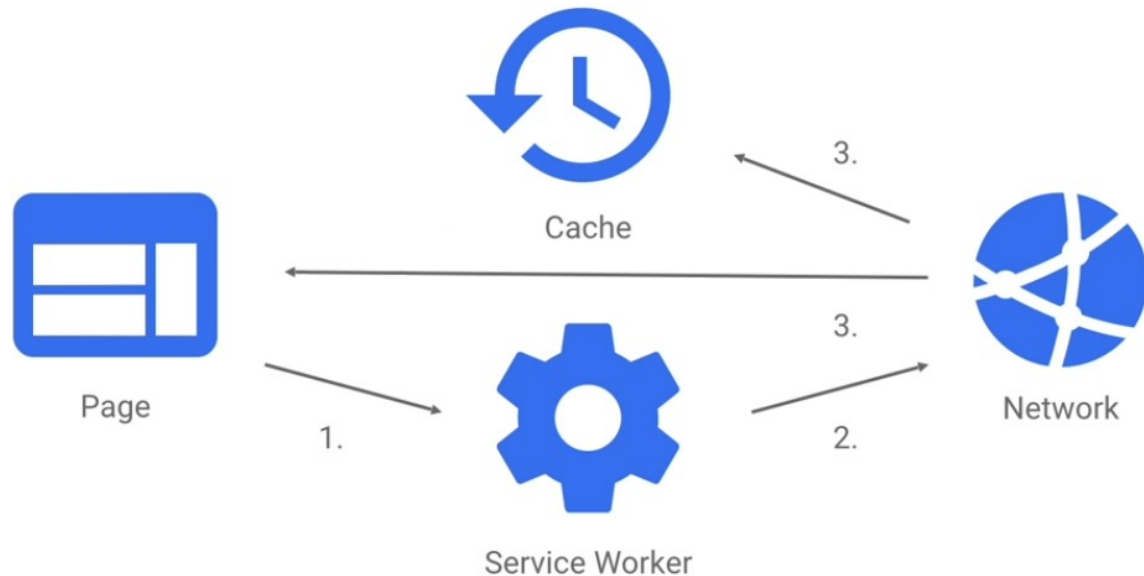
Sync

User Interaktion



```
document.querySelector('.cache-article')
  .addEventListener('click', event => {
    event.preventDefault();
    const id = this.dataset.articleId;
    caches.open('mysite-article-' + id).then(cache => {
      fetch('/get-article-urls?id=' + id).then(response => {
        return response.json();
      }).then(urls => {
        cache.addAll(urls);
      });
    });
  });
```

Online Status – Cache then Network



```
self.addEventListener('fetch', event => {  
  event.respondWith(  
    caches.open('mysite-dynamic').then(cache => {  
      return cache.match(event.request).then(response => {  
        return response || fetch(event.request)  
      }).then(response => {  
        cache.put(event.request, response.clone());  
        return response;  
      });  
    });  
  });  
});
```

Live Coding Projekt 3

Übung 2

Aufgabe:

Erstellen Sie ein beliebiges Projekt mit einem Service Worker. Überlegen Sie sich eine Caching Strategie und wenden Sie diese in Ihrem Service Worker für die Fetch API an! Überlegen Sie zusätzlich, welche Dateien in welchem Cache gespeichert werden sollen!

Gerne können Sie das Projekt aus Übung 1 benutzen!

Hinweise:

Sie sind in der Auswahl der Frameworks/Sprachen nicht eingeschränkt!

Gerne können Sie auch Workbox benutzen!

Verwenden Sie den Offline Modus in den Dev Tools!

Zeit: 45 Minuten

Push Notification

Push Notification sollen eine Eigenschaft wie in Nativen Apps bieten

Benötigt die Zustimmung des Benutzers

Über eine bestimmte Subscription erhält der Benutzer eine Nachricht

Dies ist auf allen Geräten möglich

Möglichkeit mit DOM Notification API Methoden

Möglichkeit mit „push“ Service Worker Event

VAPID Keys erhöhen die Sicherheit

Möglichkeit der Speicherung in einer Datenbank

Jeder Browser hat eigene Subscription URL

Push Notification

Kompatibilität:

	Android (Chrome)	Android (Firefox)	iOS (Safari)	Desktop (Chrome/Edge)	Desktop (Firefox)
Push-Notifications	Ja	Ja	Nein	Teilweise	Teilweise

Größter Nachteil an Push Notifications ist der Browser Support.

Zusätzlich werden von JEDEN Browser Push Notification in privaten und incognito Modi verboten

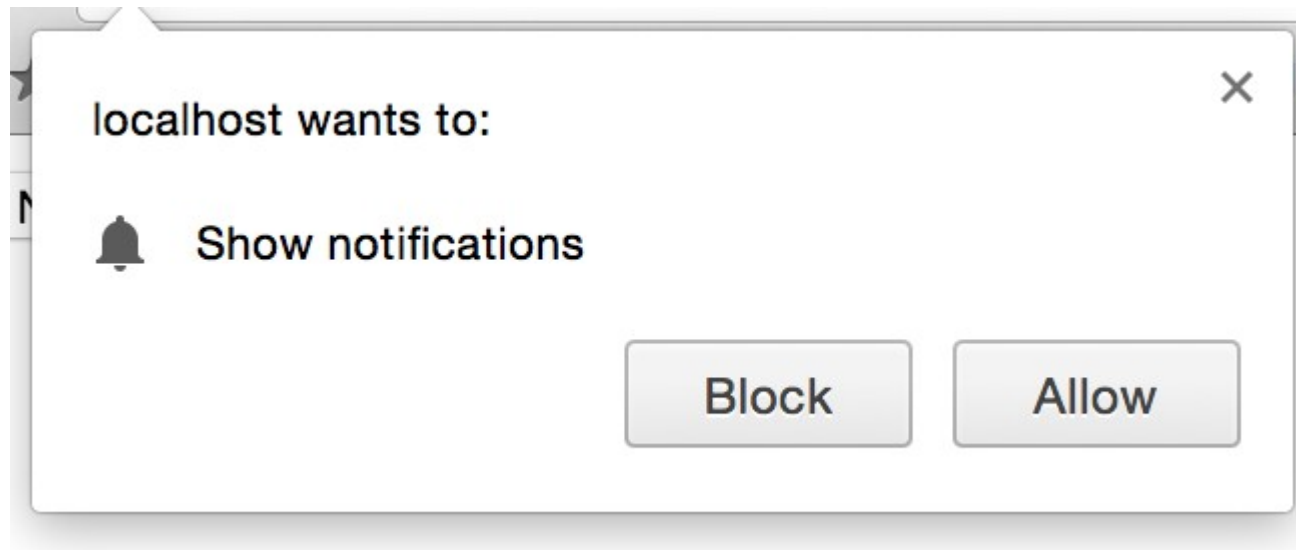
Push Notification

Zustimmung:

Zustimmung muss für jede Domain vom Nutzer bestätigt werden

Einmal erteilt, bleibt diese bis auf Widerruf erhalten

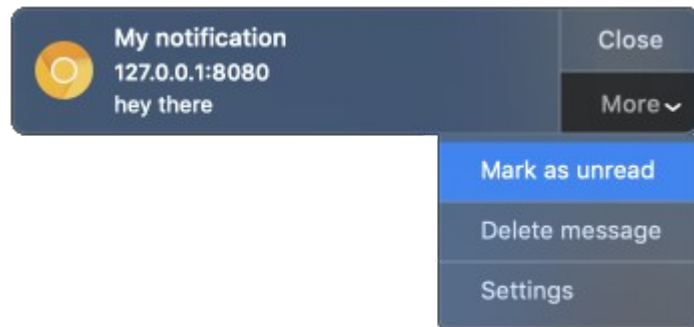
Die Zustimmung wird gleich wie Standort, Kontakte, File System abgefragt



Push Notification

Zustimmung:

Nach erfolgreicher Zustimmung können diese erhalten werden



Die Inhalte einer Benachrichtigung enthalten Informationen wie:

Titel, Text, Icon, Vibration, Action Buttons und Domainnamen

Diese werden durch das Notification Settings Object befüllt

Push Notification

DOM Notification API Methoden:

Das Standard Notification Object mit Methoden kommt innerhalb des DOM's für jeden Browser

Bereits diese werden unterschiedlich unterstützt

Auf dieses Notification Object können wir im „push“ Service Worker Event zugreifen

Das Layout ist vorgegeben

Ausgabe erfolgt über „new“ Konstruktor oder über showNotification Methode

Push Notification

	📱					📱					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
Notification	📌 20 *	✓ 14	✓ 22 ...	✓ 23 *	✓ 7	📌 42	📌 22 ...	📌 29	✗ No	📌 4.0	✗ No
Notification() constructor	✓ 20	✓ 14	✓ 22 ...	✓ 23	✓ 7	✗ No	✓ 22 ...	✗ No	✗ No	✗ No	✗ No
actions 🧪	✓ 53	✓ 18	✗ No	✓ 39	✗ No	✓ 53	✗ No	✓ 41	✗ No	✓ 6.0	✗ No

```

async function showNotification() {
  const result = await Notification.requestPermission();
  if (result === 'granted') {
    const noti = new Notification('Hello!', {
      body: 'It's me.',
      icon: 'mario.png'
    });
    noti.onclick = () => alert('clicked');
  }
}
showNotification();

```

Push Notification

Service Worker:

Durch den Service Worker kann auf das Event „push“ reagiert werden

Diese Push API kümmert sich um Events, welche vom Server erhalten werden

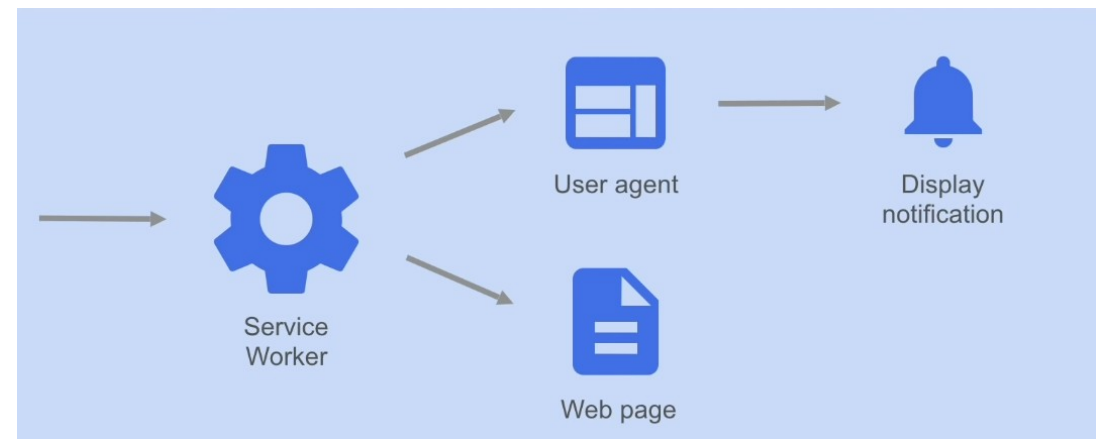
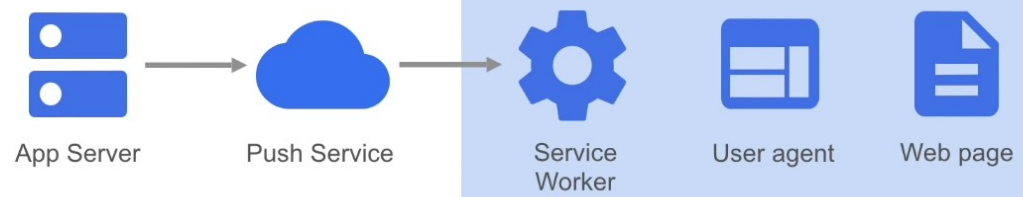
Weiterführend:

Im Service Worker können wir auf `notificationclose` und `notificationclick` als Event zugreifen

Push Notification

Service Worker Subscription:

1. Der Benutzer registriert sich im Browser zu einen Browserendpunkt
2. Diese Daten werden im Server gespeichert
3. Der Server sendet eine Push Benachrichtigung, auf dessen sich die Benutzer registriert haben
4. Der Browser mit dem Service Worker reagiert auf diese Push Benachrichtigung



Push Notification

Service Worker:

Die Registrierung kann ebenfalls durch den Service Worker durchgeführt werden

Das Subscription Object erhält der Code von `pushManager.subscribe()`

```
navigator.serviceWorker.getRegistration()  
.then(function(reg) {  
  reg.pushManager.subscribe({  
    userVisibleOnly: true  
  }).then(function(sub) {  
    // send sub.toJSON() to server  
  });  
});
```

Push Notification

Service Worker:

Das Subscription Object enthält immer den Endpunkt vom Browser und den verschlüsselten publicKey aus dem für den p256dh und dem privateKey für die verschlüsselte Messaging Kommunikation

```
{  
  "endpoint":  
    "https://android.googleapis.com/gcm/send/  
f1LsxxKp...",  
  "keys": {  
    "p256dh": "BLc4xRzK1KORKW1b0QRv-1n...",  
    "auth": "5I2Bu2oKdyy9CwL8QVF0NQ=="  
  }  
}
```


Push Notification

Service Worker:

Der Server muss das Subscription Objekt verarbeiten, gegebenenfalls speichern, und danach eine Nachricht zu versenden als Antwort auf die Anfrage

Dies basiert auf dem Packet web-push

```
var webPush = require('web-push');
var payload = 'Here is a payload!';
var options = {
  gcmAPIKey:
    'AIzaSyBVImB3hJJ...8J5D4xnFo2fFI',
  TTL: 60
};
webPush.sendNotification(pushSubscription,
  payload, options);
```

Push Notification

VAPID Keys:

Voluntary Application Server Identification for Web Push (VAPID)

Dient zur Identifizierung der Subscription

VAPI benutzt JSON Web Token (JWT)

Dient zur Authentifizierung des Benutzers in der Registrierung und für den Server basierend auf der Registrierung

Besteht aus publicKey und privateKey

Beim senden muss das publicKey von urlBase64 zu Uint8Array umgewandelt werden

Die Erstellung dieser Keys sollte einmalig auf dem Server durchgeführt und gespeichert werden

Push Notification

VAPID Keys:

```
var webPush = require('web-push');
var payload = 'Here is a payload!';
var options = {
  vapidDetails: {
    subject: 'mailto:
      example-email@example.com',
    publicKey: vapidPublicKey,
    privateKey: vapidPrivateKey
  }
};
webPush.sendNotification(pushSubscription,
  payload, options);
```

Push Notification

Service Worker Client:

1. Push Event erscheint im Browser
2. Service Worker ist aktiv
3. Service Worker Event „push“ bearbeitet die Nachricht

```
self.addEventListener('push', function(e) {  
  var title = e.data.text();  
  e.waitUntil(  
    self.registration.showNotification(title)  
  );  
});
```

Live Coding Projekt 4

Live Coding Projekt 5

Übung 3

Aufgabe:

Erstellen Sie ein beliebiges Projekt mit einem Service Worker. Holen Sie sich vom Benutzer die Einwilligung, dass Push Notifications geschickt werden dürfen! Lassen Sie den Benutzer in seinen Browser registrieren und schicken Sie diese Information an einen Server, welcher eine Push Nachricht schickt. Verarbeiten Sie diese Push Nachricht im Server Worker mit dem Event!

Hilfe beim Server: <https://github.com/Rambarz/pwa>

Hinweise:

Sie sind in der Auswahl der Frameworks/Sprachen nicht eingeschränkt!

Gerne können Sie auch web-push und VAPID Keys benutzen!

Bedenken Sie den Browser Support oder vergewissern Sie sich dessen!

Zeit: 45 Minuten

Background Sync

Background Sync oder allgemein Sync beschreibt ein Service Worker Event

Service Worker Event „sync“ wird unter Offline/Online ausgelöst

Die Registrierung des Syncs kann auch bei Offline Status durchgeführt werden

Codefreigabe nach dem Ereignis kann alles beinhalten

Innerhalb eines Sync kann z.B. eine API angesprochen werden oder Cache

Großes Problem: Kompatibilität

Background Sync

Kompatibilität:

	🖥️					📱					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
<code>sync</code> 🧪	✓ 49	✓ 79	✗ No	✓ 36	✗ No	✓ 49	✗ No	✓ 36	✗ No	✓ 4.0	✓ 49

	🖥️					📱			
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS
<code>SyncManager</code> 🧪	✓ 49	✓ 79	✗ No	✓ 36	✗ No	✓ 49	✗ No	✓ 36	✗ No
<code>getTags</code> 🧪	✓ 49	✓ 79	✗ No	✓ 36	✗ No	✓ 49	✗ No	✓ 36	✗ No
<code>register</code> 🧪	✓ 49	✓ 79	✗ No	✓ 36	✗ No	✓ 49	✗ No	✓ 36	✗ No
Available in workers 🧪 ⚠️	✓ 61 ...	✓ 79	✗ No	✗ No	✗ No	✓ 61 ...	✗ No	✗ No	✗ No

Background Sync

Großes Problem: Kompatibilität

Überprüfung auf `registration.sync` für Unterstützung

Jeder Sync hat einen bestimmten TAG Namen

Jeder Sync MUSS registriert werden

Dieser TAG kann bestimmten Code ausführen, dieser MUSS ein Promise zurückgeben

Sobald der Service Worker im Hintergrund Online ist, triggert er den TAG

Offline Verhalten zum laden von Daten, sobald der Benutzer wieder Online ist

Einsatz: Emails, News, Filme/Musik

Background Sync

```
navigator.serviceWorker.ready.then(registration => {
  if (registration.sync) {
    // Background Sync is supported.
  } else {
    // Background Sync isn't supported.
  }
});

async function requestBackgroundSync() {
  await self.registration.sync.register('my-tag-name');
}

self.addEventListener('sync', event => {
  if (event.tag === 'my-tag-name') {
    event.waitUntil(doTheWork());
  }
});
```

Beispiel: <https://quirky-rosalind-ac1e65.netlify.app/>

Live Coding Projekt 6

Übung 4

Aufgabe:

Erstellen Sie ein beliebiges Projekt mit einem Service Worker. Registriere Sie ein SYNC Event und ein TAG! Verarbeiten Sie das SYNC Event und überprüfen Sie auf die TAG Namen und führen Sie auf diesen einen Promise aus z.B. `fetch()`! Gehen Sie offline und online für das Auslösen des SYNC Events im Service Worker!

Hinweise:

Sie sind in der Auswahl der Frameworks/Sprachen nicht eingeschränkt!

Nutzen Sie die Entwickler Dev Tools für das Debugging!

Bedenken Sie den Browser Support oder vergewissern Sie sich dessen z.B. Firefox!

Zeit: 45 Minuten

File System

Service Worker oder PWA generell biete keine eigene Implementation für das bearbeiten/behandeln von Dateien und Ordner

Es gibt kein eigenes Event/Methode diesbezüglich

Verbindung mit PWA natürlich möglich

Grundlage bleibt JavaScript für den Zugriff auf das File System

Lösung: File System API des DOM

File System

Kompatibilität:

	🖥️					📱					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
<code>FileSystemHandle</code>	✓ 86	✓ 86	✗ No	✓ 72	✓ 15.2	✓ 86	✗ No	✗ No	✓ 15.2	✓ 14.0	✗ No
<code>isSameEntry</code>	✓ 86	✓ 86	✗ No	✓ 72	✓ 15.2	✓ 86	✗ No	✗ No	✓ 15.2	✓ 14.0	✗ No
<code>kind</code>	✓ 86	✓ 86	✗ No	✓ 72	✓ 15.2	✓ 86	✗ No	✗ No	✓ 15.2	✓ 14.0	✗ No
<code>name</code>	✓ 86	✓ 86	✗ No	✓ 72	✓ 15.2	✓ 86	✗ No	✗ No	✓ 15.2	✓ 14.0	✗ No
<code>queryPermission</code> ⚠️	✓ 86	✓ 86	✗ No	✓ 72	✗ No	✓ 86	✗ No	✗ No	✗ No	✓ 14.0	✗ No
<code>requestPermission</code> ⚠️	✓ 86	✓ 86	✗ No	✓ 72	✗ No	✓ 86	✗ No	✗ No	✗ No	✓ 14.0	✗ No

Beispiel: <https://whatpwacando.today/file-system>

IE	Low	▼
Edge	High	▼
Firefox	Low	▼
Chrome	Medium	▼
Safari	Low	▼
Opera	Medium	▼
Safari on iOS	Medium	▼
Android Browser	Low	▼
Opera Mobile	Low	▼
Chrome for Android	Low	▼
Firefox for Android	Low	▼
Samsung Internet	Low	▼

Gliederung

7. Weiterführungen

Weiterführungen

Authentifizierung und Autorisierung

Contact Picker, etc.

IndexedDB

Performance Verbesserung

NativeFramworks versus PWA

Authentifizierung und Autorisierung

Ansatz der Authentifizierung und Identifizierung bleibt bestehen

Grundprobleme mit Authentifizierung und Login/Register

Name – Passwort Kombination

Entwickler Aufgaben für Verschlüsselung, Plain Text, Strong Password Police, 2FA

Zugriffe auf window Objekt im JavaScript bleibt bestehen

PWA's benötigen dieses window Objekt für die Registrierung

Service Worker laufen immer in einen extra Thread

Programmiersprache bleibt JavaScript (TypeScript) und wird vom Browser interpretiert

Contact Picker etc.

PWA erweitern die Web Application

Weitere angekündigte PWA Features wie Contact Picker, Bluetooth, Audio, Standorte und NFC sind ebenfalls wie das „sync“ Event stark von Browser Kompatibilität betroffen

PWA greift nicht direkt in einen Quellcode ein oder fügt JavaScript Code hinzu

Die jeweiligen Browser API Schnittstellen sind nicht von PWA abhängig

Szenarien:

Contact Picker API, File System API, Web Bluetooth, Web-NFC, IndexedDB

Beispiel: <https://whatpwacando.today/bluetooth>

IndexedDB

IndexedDB :

Frontend NoSQL Datenbank – nicht relational – kein SQL

Hält JavaScript Objekte nach einem Key-Ansatz wie Redis(Objecte, Dateien, Blobs)

IndexedDB besteht aus einem ObjectStore für die Daten

#	Key (key-path 'name')	Value
0	John Lennon	{name: John Lennon, nickname: 'The smart one', age: 40, living: 'no'}
1	Paul McCartney	{name: Paul McCartney, nickname: 'The cute one', age: 73, living: 'yes'}
2	George Harrison	{name: George Harrison, nickname: 'The shy one', age: 58, living: 'no'}
3	Ringo Starr	{name: Ringo Starr, nickname: 'The funny one', age: 74, living: 'yes'}

IndexedDB

IndexedDB API:

Häufige Probleme mit der IndexedDB API bezüglich Funktionalität

Alte asynchrone API mit callbacks und Events auslöst

Versionierung mit Upgradable

Häufig genutzt um in PWA große Dateien zu speichern (Bilder, Dateien)

Unterstützt CRUD Operatoren (add-addAll, get, put, delete)

Kann im DOM Object window untersucht werden

Lösungsansatz: IndexedDB Promised (von Google)

Beispiele folgen...

IndexedDB

Beispiel Datenbank Erstellung:

```
var dbPromise = idb.open('test-db', 1,
function(upgradeDb) {
  if (!upgradeDb.objectStoreNames
    .contains('store')) {
    upgradeDb.createObjectStore('store');
  }
});
```

- `upgradeDb.createObjectStore('people', {keyPath: 'email'});`
- `upgradeDb.createObjectStore('notes', {autoIncrement:true});`
- `upgradeDb.createObjectStore('logs', {keyPath: 'id', autoIncrement:true});`

IndexedDB

Beispiel Index Erstellung für Key Zugriff:

```
var dbPromise = idb.open('test-db', 1,
function(upgradeDb) {
  if (!upgradeDb.objectStoreNames
    .contains('ppl')) {
    var store = upgradeDb.createObjectStore('ppl')
    store.createIndex('email', 'email',
      {unique: true});
  }
});
```

IndexedDB

Beispielanfrage Create(add) und Read(get):

```
dbPromise.then(function(db) {  
  var transaction = db.transaction('ppl',  
    'readwrite');  
  var store = transaction.objectStore('ppl');  
  store.add({name: 'Fred'});  
  return transaction.complete;  
})
```

```
dbPromise.then(function(db) {  
  var tx = db.transaction('ppl', 'readonly');  
  var store = tx.objectStore('ppl');  
  return store.get('Fred');  
});
```


IndexedDB

Beispielanfrage Update(put) und Delete(delete):

```
dbPromise.then(function(db) {  
  var tx = db.transaction('store', 'readwrite');  
  var store = tx.objectStore('store');  
  var item = {name: 'Fred', email:  
    'fred@example.com'}  
  store.put(item);  
  return tx.complete;  
});
```

```
dbPromise.then(function(db) {  
  var tx = db.transaction('ppl', 'readwrite');  
  var store = tx.objectStore('ppl');  
  store.delete('Fred');  
  return tx.complete;  
});
```

Performance Verbesserung

Performance gebunden an Web Performance und Architektur

Web Performance Verbesserungen:

Bundle Analyse und Bundle Größe

Web Font

Bilder

Minimierte Dateien

Routing Module

Fehlerreduzierung, z.B. try-catch, Netzwerke, API's, Error Handling

Performance Verbesserung

PWA Performance Verbesserungen:

Caching

Lighthouse → späteres Kapitel

Lazy Loading

Apple HTML Head Support

PRPL Strategie

Performance Verbesserung

PRPL Strategie:

Preload – Render – PreCache – Lazy Loading

Preload: Wichtigste Informationen preloaden, z.B. CSS und JS Module

Beispiel: `<link rel="preload" as="style" href="style.css">`

Render: Möglichst schnelle First-Interaction durch bekannte Verbesserungen

Beispiel: Aufschieben von nicht kritischen Inhalten durch async JS oder external CSS

PreCache: Gewisse Dateien sollen im Cache für erneute Abfragen gespeichert werden

Beispiel: Service Worker!

Lazy Loading: Verschiedene Ansätze für das Laden von Ressourcen, erst wenn diese benötigt werden

Beispiele: img TAG Element Attribut loading, Preload CSS+Fonts, JS Module, CSS Media

NativeFrameworks versus PWA

Frameworks zur Erstellung von möglichst Plattform/OS unabhängigen Anwendungen gewinnen an Beliebtheit

Oft wird auf ein bestehendes Prinzip aufgebaut

Beispiele: JavaScript, React, Swift, Vue

Die dazugehörigen Lösungen: Flutter, Ionic, React Native, Swift, Vue CLI

Diese sind aber nicht mit dem Ziel PWA entstanden, sondern meist mit dem Ziel Native Apps

Zur Erinnerung: PWA basieren auf Web Application und sollen den Zugang jeden möglich machen

Gegenüberstellungen...

NativeFrameworks versus PWA

„PWA basieren auf Web Application und sollen den Zugang jeden möglich machen“

Jede dieser Technologien lassen sich mit PWA's erweitern

Dabei gibt es Unterschiede bezüglich der Einstiegshürde und der Unterstützung

Ionic: Toolkit als Codebasis mit JavaScript für unabhängige Applicationen

React Native: React Erweiterung für Native App mit Render Cycle

Flutter: Toolkit als Codebasis mit Dart Ansatz (Java, JS, Python) für unabhängige Applicationen

Mögliches Szenario: JavaScript Stack mit: ReactJS + Ionic + PWA + Workbox

Fazit: ALLE Frameworks/Libraries/Toolkits/HTML unterstützen PWA

Gliederung

8. Registrierung

Registrierung

Registrierung in App Stores kann die Akzeptanz erhöhen

Oftmals haben Unternehmen allerdings bereits eine Native App

Problem bei Apple bezüglich Manifest Akzeptierung

Google bietet die Möglichkeit eine TWA zu registrieren

Google Play Store, Samsung Galaxy Store und Microsoft Store

Bedingung: Android Studio

Registrierung

TWA:

Trustet Web Application

Muss mit Android Studio eine Java Unterstützung aktivieren für eine APK Datei

Registrierung durch eine Support Bibliothek grundle

Aktivierung durch eine Androd Manifest it dem URL Link

SHA 256 Schlüssel wird von dem URL Link zur APK benötigt

Weitere Information werden in AndroidManifest.xml ähnlich wie manifest.jsen gespeichert

Erstellung des App Bundles und Hochladen in Store mit Developer Account

Frage: Möchte ein Unternehmen das?

Registrierung

Möchte ein Unternehmen das?

Nur Native Apps müssen über die App Store die Richtlinien einhalten

Downloadgröße ist wieder zwischen 10MB – 80MB statt Browser

Provision von 15% - 30% durch In-App-Käufe

Aufwendige App Updates

Zusätzliche Arbeit

Registrierung

Desktop:

Registrierung auf dem Desktop ist mit Hilfe eines Browsers zu erfüllen

Der Browser erkennt die PWA und bietet eine Installation/Link an

Google Chrome erkennt dies automatisch und fragt den Benutzer

Alternativ kann der Nutzer auf die Installation hingewiesen werden

Möglichkeit das Programm per Taskleiste oder Suche auszuführen

Gliederung

9. Testing

Testing

Lokales Testen wie in einer DEV Umgebung sollte in Chrome durchgeführt werden

Zugang durch Dev Tools

Support Abdeckung sollte aus aktuellen Quellen bezogen werden (MDN, Google Developer)

Testabdeckung durch Endgeräte kann durch USB Debugging durchgeführt werden

Testabdeckung durch Browser sollte durch manuelles Eingreifen durchgeführt werden

Software Testabdeckung KANN durch Tools wie z.B. JEST Cache Test absolviert werden

PWA Eigenschaften und Merkmale nur durch Lighthouse abdeckbar

Lighthouse ist ebenfalls nur in Chrome vorhanden, quasi Live

Lösung: Lighthouse CLI

Testing

4 Merkmale sollten IMMER vorhanden sein
Es existierte eine Checkliste für Google Devs
Neu: web.dev

Lighthouse stellt Bewertung aus
Diese Merkmale sind bekannt

Baseline Progressive Web App Checklist

The [Lighthouse tool](#) is able to automatically verify many items on the this list and may prove helpful in easily testing sites.

Site is served over HTTPS

To Test	Use Lighthouse to verify Served over HTTPS
To Fix	Implement HTTPS and check out letsencrypt.org to get started.

Pages are responsive on tablets & mobile devices

To Test	<ul style="list-style-type: none">• Use Lighthouse to verify Yes to all of Design is mobile-friendly , although manually checking can also be helpful.• Check the Mobile Friendly Test
To Fix	Look at implementing a responsive design , or adaptively serving a viewport-friendly site.

All app URLs load while offline

To Test	Load various pages in the PWA with an airplane mode enabled. Ensure the app presents some content even when offline. Use Lighthouse to verify the start URL responds with a 200 when offline.
To Fix	Use a Service Worker .

Metadata provided for Add to Home screen

To Test	Use Lighthouse to verify User can be prompted to Add to Home screen is all Yes.
To Fix	Add a Web App Manifest file to your project.

Testing

Lighthouse PWA Bewertung (Werte aus 2021):

- Ein Service Worker sollte aktiv sein
- Ein 200 Status Code sollte bei offline Modus zurück gegeben werden
- Inhalt, auch wenn JavaScript nicht verfügbar ist
- HTTPS für die neue Platform, API's und für Sicherheit
- Notfalls Weiterleitung von HTTP an HTTPS (301)
- Time to Interact muss kleiner als 10 Sekunden sein (3G Throttling)

Ergebnis: 100% Lighthouse Bewertung

Gliederung

10. Ausblick

Ausblick

1. AnyFrameworkToPick to PWA
2. Support
3. Apple's Hoffnung
4. Zusammenfassung

AnyFrameworkToPick to PWA

Konvertierung von FrontendJS

Frontend JavaScript Vue, Angular, React, Svelte, Django, Flask

Hilfe von <https://www.pwabuilder.com/>

Grundlegend kann JEDE Web Application zu einer einfachen PWA werden

Unterstützung von einigen Frameworks durch vordefinierte manifest Datei

SSR Unterstützung durch Auslieferung von den 4 PWA Merkmalen: Service Worker, Manifest, Responsive Darstellung und HTTPS

Was ist die Zielsetzung des Projektes?

Support

Support von PWA's, Service Worker, Service Worker Events, Web API'S, Browser API's

Schwieriges und komplexes Thema

Erfahrung vom Workshops und aus der Praxis zeigen große Probleme

Lücken nahezu in allen Bereichen und Anbietern/Geräten

Möchte Apple, Google, (Microsoft) und Mozilla dies?

Support

Aktuelle Tabelle von Juni 2022
(unter Vorbehalt).

	Android (Chrome)	Android (Firefox)	iOS (Safari)	Desktop (Chrome/Edge)	Desktop (Firefox)
Offline-Modus	Ja	Ja	Ja	Ja	Ja
Hintergrund-Synchronisierung von Daten	Ja	Nein	Nein	Ja	Nein
Web Share (Inhalte teilen)	Ja	Ja	Ja	Teilweise	Nein
Push-Notifications	Ja	Ja	Nein	Teilweise	Teilweise
Lokale Notifications	Nein	Nein	Nein	Nein	Nein
Download über App-Store	Ja	N/A	Nein	Nein	Nein
Hinzufügen zum Home-Screen	Ja	Ja	Nur manuell	Ja	Nein
Bluetooth	Ja	Nein	Nein	Ja	Nein
NFC	Ja	Nein	Nein	Nein	Nein
Biometrische Authentifizierung (z.B. Fingerabdruck, FaceID)	Ja	Nein	Ja	Ja	Ja
Zugriff auf Audio- und Video	Ja	Ja	Teilweise	Ja	Ja
Audio- und Videoaufnahmen	Ja	Ja	Ja	Ja	Ja
Vibration	Ja	Ja	Nein	Ja	Ja
Zugriff auf Kontaktbuch	Ja	Nein	Nein	Nein	Nein
Standortzugriff	Ja	Ja	Ja	Ja	Ja
Lage- und Beschleunigungssensor	Ja	Ja	Ja	Ja	Ja

Apple's Hoffnung

Updates zu 2023 kommenden Apple Software Update und andauernden kleinen Anpassungen und Bug Behebungen

Große Ankündigung im Juli 2022

- Push Notification
- Background Sync
- Storage (Files API)
- getUserMedia für DOM Web API's
- Web App Manifest and Standard Icons
- Add to Home Screen
- Safari Support

Zusammenfassung

Gartner Vorhersage?

Browser Herausgeber?

Benutzer Akzeptanz?

Native App Entwicklung?

Apple?

Service Worker?

Fragen?

Quellen

Bilder:

ionicframework.com, Zugriff am 15.08.2022

lumind-solutions.com, Zugriff am 20.08.2022

Medium.com, Zugriff am 20.08.2022

Cleverroad.com, Zugriff am 20.08.2022

Builtvisible.com, Zugriff am 18.08.2022

Prestashop.com, Zugriff am 17.08.2022

Statista.com, Zugriff am 20.08.2022

Gartner.com, Bericht vom 05.08.21, Key Considerations when building apps, Zugriff am 20.08.2022

Google Chrome Developers Network, Zugriff am 20.08.2022

Google Chrome Developers YouTube Channel, Zugriff am 20.08.2022

lonos.de, Zugriff am 21.08.2022

Medium.com, Zugriff am 21.08.2022

Eigene Darstellung, erstellt mit Photoshop und Paint, Erstellung zwischen 17.08.2022 und 21.08.2022

Texte:

Progressive Web Apps with React, Ausgabe Packt 2018

Google Chrome Developers YouTube Channel, Zugriff am 20.08.2022

Informationen:

Fireshop.io, Zugriff am 16.08.2022

Progressive Web Apps with React, Ausgabe Packt 2018

Freecodecamp.com, Zugriff am 15.08.2022

eCommerce-vision.de, Zugriff am 16.08.2022

dcode, Zugriff am 15.08.2022

Web.dev, Zugriff am 16.08.2022

Developers.google.com, Zugriff am 18.08.2022

Codeytek.com, Zugriff am 18.08.2022

Copyright

© Copyright an dieser
Präsentation hat der Ersteller
Florian Hahn
21.08.2022