Napredni razvoj programske potpore za web

- predavanja -

2021./2022.

Progresivne web-aplikacije

PWAs

Creative Commons

- slobodno smijete:
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo
- pod sljedećim uvjetima:
  - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava. *Tekst licence preuzet je s http://creativecommons.org/*

Progressive Enhancement

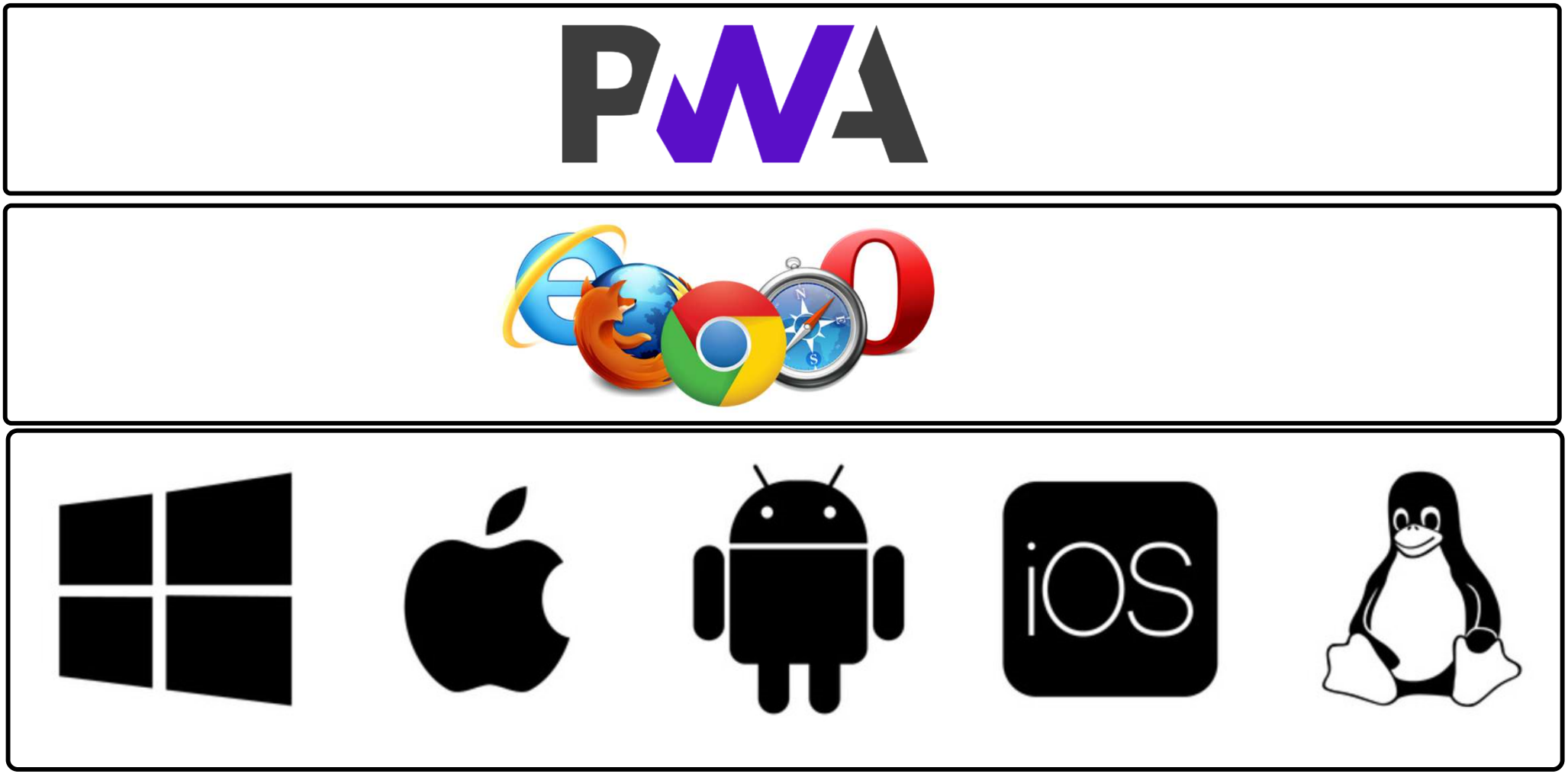
- *Progressive enhancement is a **design philosophy** that provides a **baseline of essential content** and functionality to as many users as possible, while delivering the **best** possible **experience only** to users of the most **modern browsers** that can run all the required code.* https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement
- Na starijim preglednicima svejedno uporabljivo
- Unaprjeđuje se korisničko iskustvo za korisnike sposobnijih/modernijih preglednika
- Uobičajene tehnike:
  - *Feature detection*
  - *Polyfills*
- Povezan pojam: *Graceful degradation* (isto načelo, suprotnih "smjer kretanja")

PWAs

- ***Progressive Web Apps*** *(PWAs) are web apps that use service workers, manifests, and other web-platform features in combination with progressive enhancement to give users an experience* ***on par with native apps****.* https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
- Omogućuju mnoge prednosti:
  - Installable -> Home screen
  - progressively enhanced
  - responsively designed
  - re-engageable →PUSH notifikacije
  - Linkable
  - discoverable
  - network independent -> Service workers, keširanje
  - secure

Internet preglednik!

- PWA su itekako utemeljene na preglednicima
- Preglednici poput virtualnog stroja - apstrahiraju OS!
  - Postoje neke sličnosti s Electron radnim okvirom



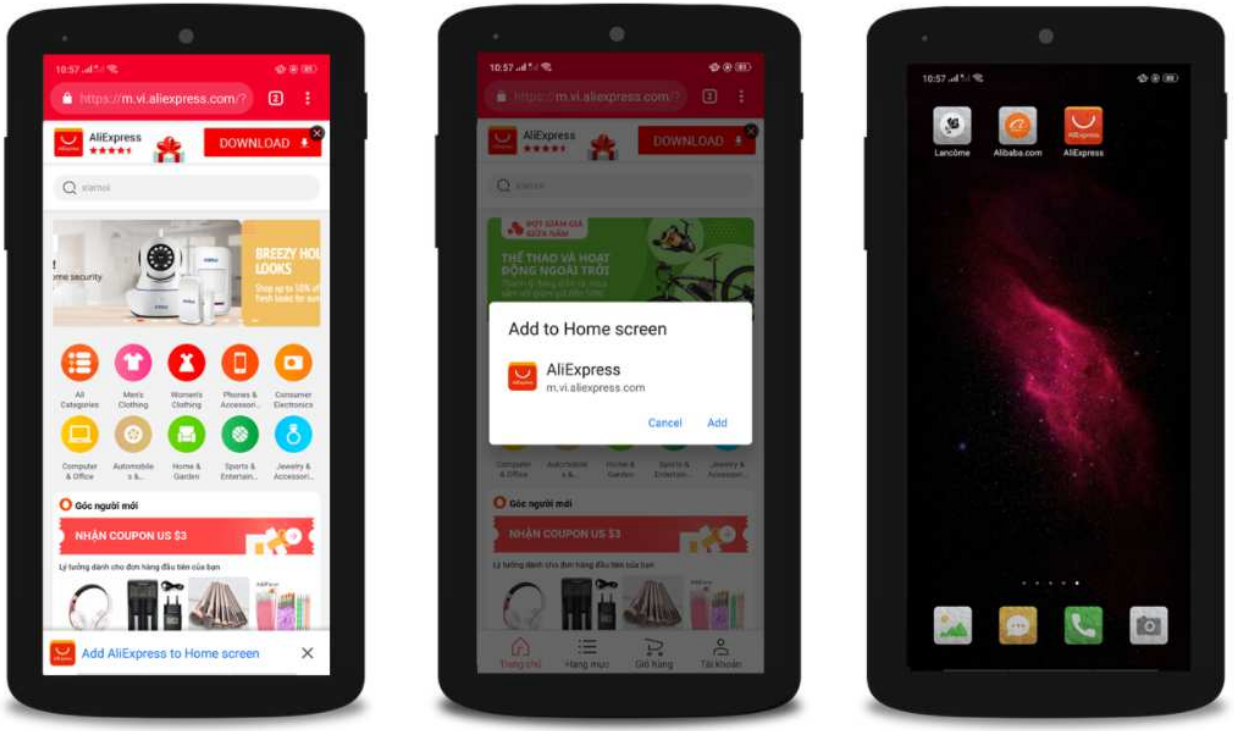
Kada i zašto, tko?

- Kada web-app možemo smatrati PWA?
  - Ne postoji jednoznačan odgovor, *Lighthouse*
- Kako ju prepoznati?
  - Teško, ali to i je ideja ☺
  - (slika na sljedećem slajdu)

- Tko koristi?
  - Mnogi: https://www.tigren.com/examples-progressive-web-apps-pwa/
- Je li 100% prihvaćeno
  - Ne - Apple, Firefox, …

Kako prepoznati?

- Npr., aliexpress



PWA tehnologije

1. **Manifest**
2. **Service worker**
   - *Caching*
   - *Offline* rad
   - *Background sync*
   - *Push* notifikacije

- Često u kombinaciji s:
  - Media API
    - Kamera
    - Mikrofon
  - Geolocation API
    - Pozicija
  - Lokalna pohrana podataka
    - IndexedDB

1. Manifest (značajniji atributi)

```
{
    "name": "PWA-Cookbook-01",
    "short_name": "CookBook",
    "description": "Demo aplikacija u predmetu Napredni razvoj za web",
    "display": "fullscreen",
    "theme_color": "red",
    "background_color": "#DDD",
    "start_url": "/index.html",
    "scope": ".",
    "orientation": "portrait-primary",
    "dir": "ltr",
    "lang": "en-US",
    "icons": [
      {
        "src": "img/windows10/SmallTile.scale-100.png",
        "sizes": "71x71"
      },
      ...
```

- Standalone
- Fullscreen
- Minimal-UI
- Browser

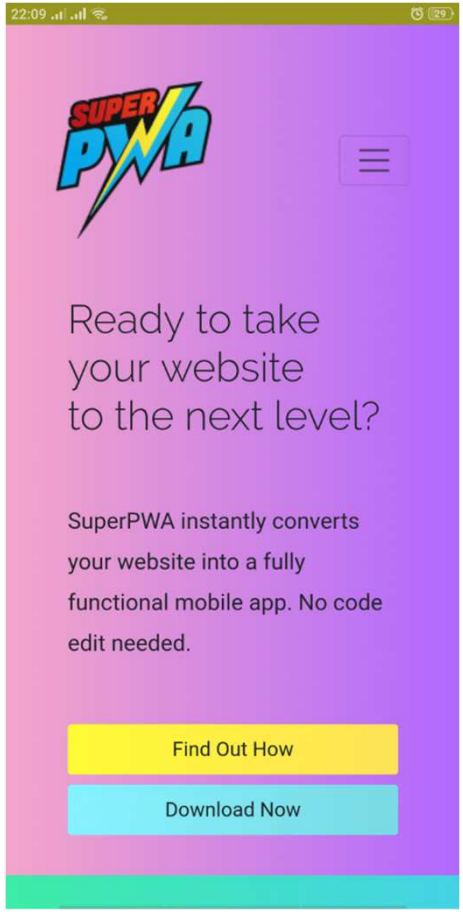
The background_color member defines a placeholder background color for the application page to display before its stylesheet is loaded. This value is used by the user agent to draw the background color of a shortcut when the manifest is available before the stylesheet has loaded.

The start_url member is a string that represents the start URL of the web application — the preferred URL that should be loaded when the user launches the web application (e.g., when the user taps on the web application's icon from a device's application menu or homescreen).

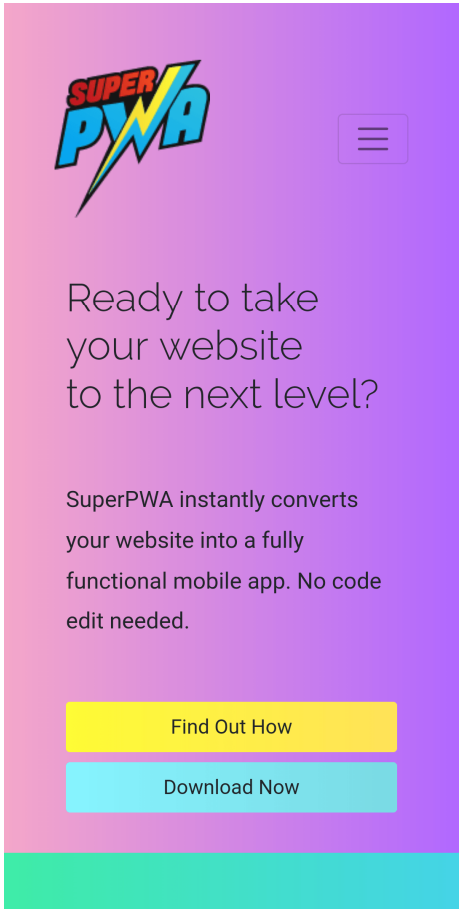
The scope member is a string that defines the navigation scope of this web application's application context. It restricts what web pages can be viewed while the manifest is applied. If the user navigates outside the scope, it reverts to a normal web page inside a browser tab or window.
If the scope is a relative URL, the base URL will be the URL of the manifest

PWA display modes

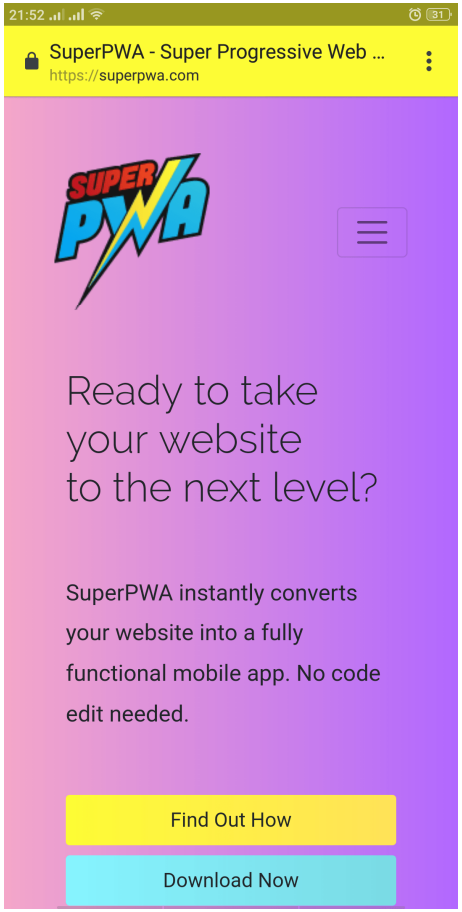
Standalone



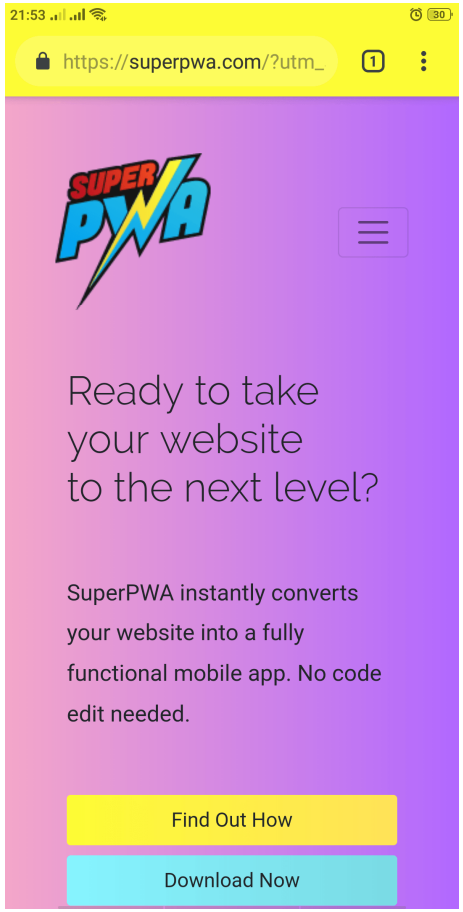
Fullscreen



Minimal-UI



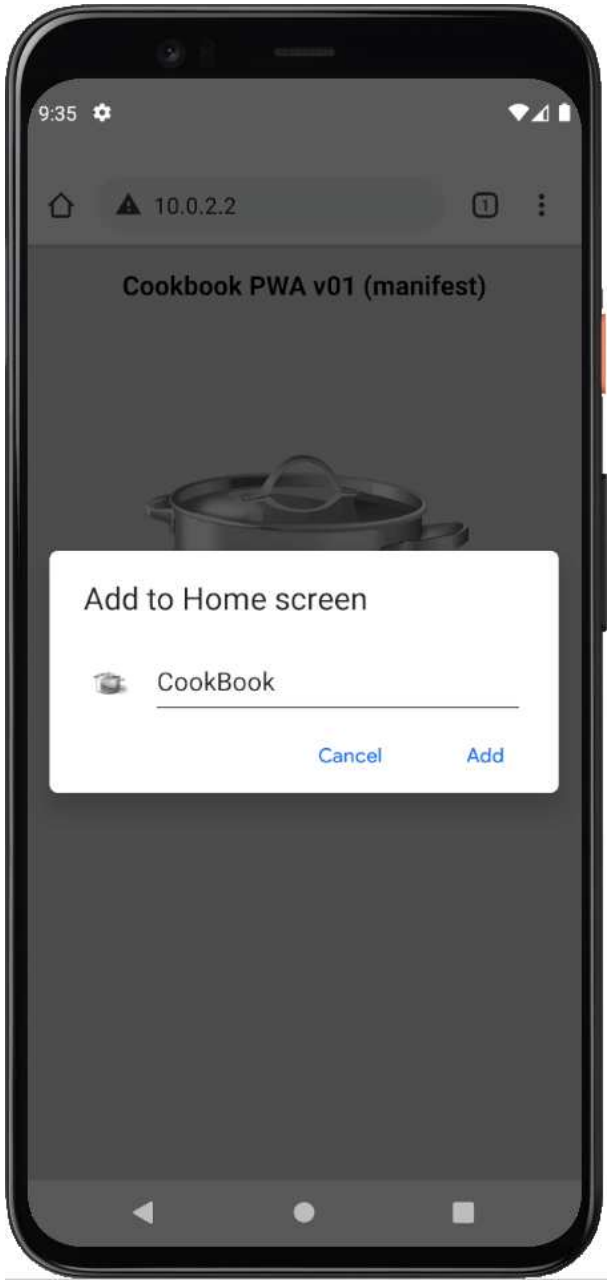
Browser



fallback

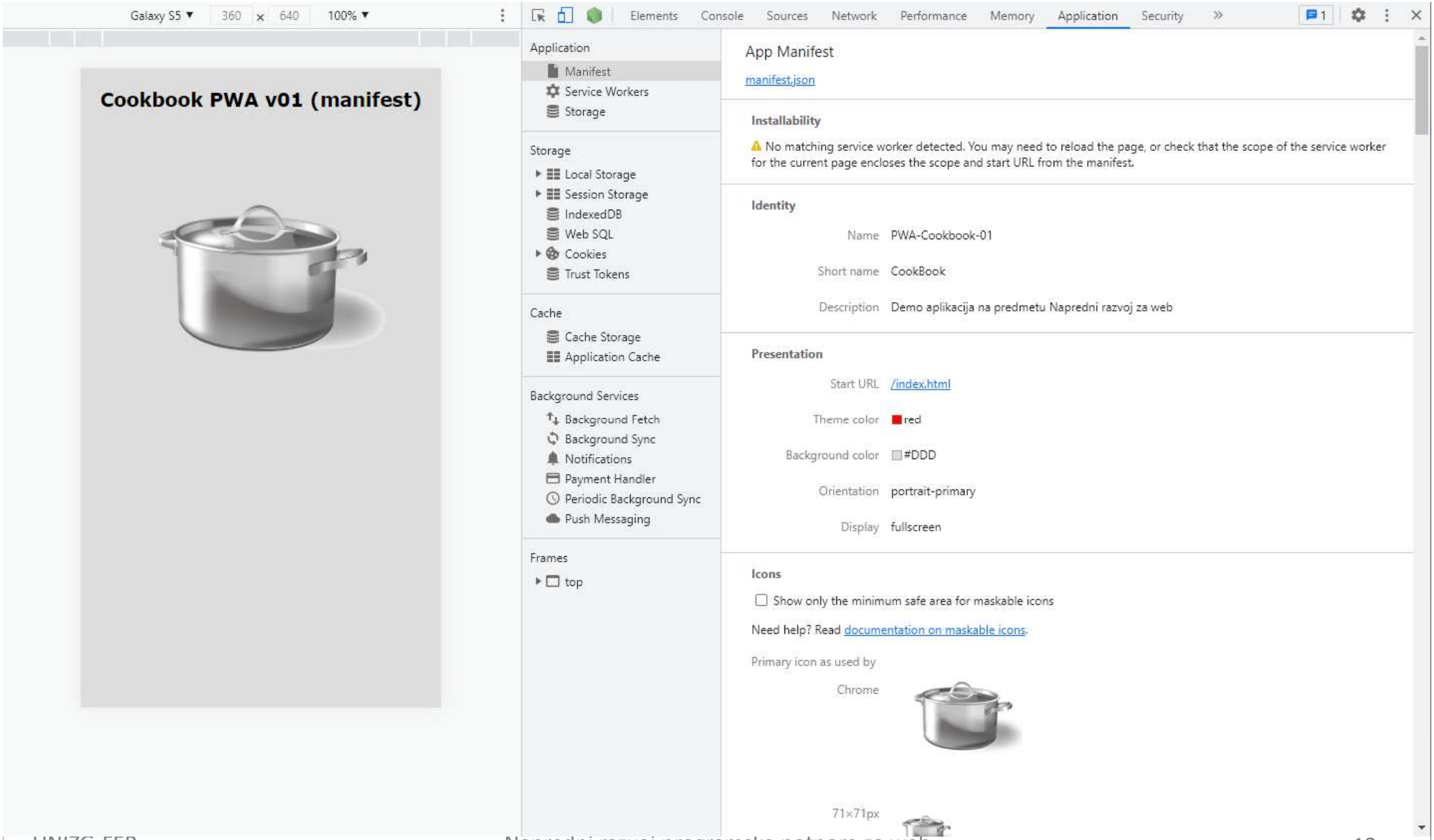
Probajmo s emulatorom

- Android studio
- Tools -> AVD manager
- Otvorimo 10.0.2.2 (a ne localhost, jer to bio sam pametni telefon)
- Chrome
  - Možemo (zasad samo) ručno dodati na Home screen



Pogledajmo i devtools -> Application

VER: 01

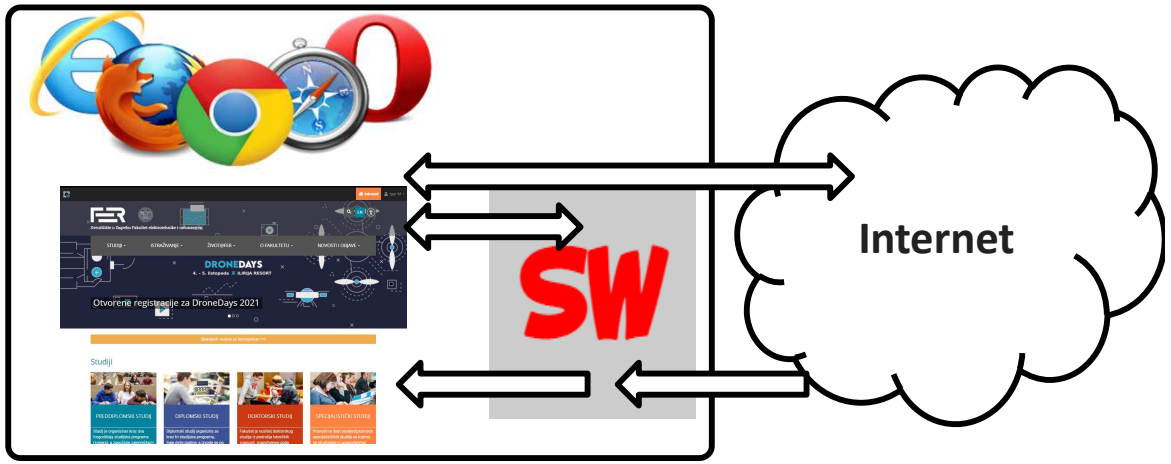


Installability

- Navodno ažurni kriteriji za Chrome: https://developers.google.com/web/fundamentals/app-install-banners/native
  - nisu
- **Dodao:**
  - prefer_related_applications, related_application
  - **HTTPS** s ispravnim certifikatom
    - Rješenje – postaviti aplikaciju na neki besplatan hosting, npr. https://www.cloudsavvyit.com/5057/how-to-host-a-static-website-for-free-on-googles-firebase-hosting-platform/
    - https://pwa02-92063.web.app/
  - Service worker, registiran, fetch handler

2. Service worker

- **Izvanmrežni** (*offline*) rad
  - Keširanje
  - Možemo presretati ***fetch*** zahtjeve i sami odgovarati
- **Push** notifikacije
- Service worker je JS datoteka
  - Naš JS kod s kojim kontroliramo ponašanje
- **Izvodi se mimo glavne preglednikove UI dretve**
  - **ServiceWorkerGlobalScope**
  - **nemaju pristup DOM-u**
  - **`self` objekt**
- **Preduvjet – HTTPS**
  - Iznimka:
    http://localhost...

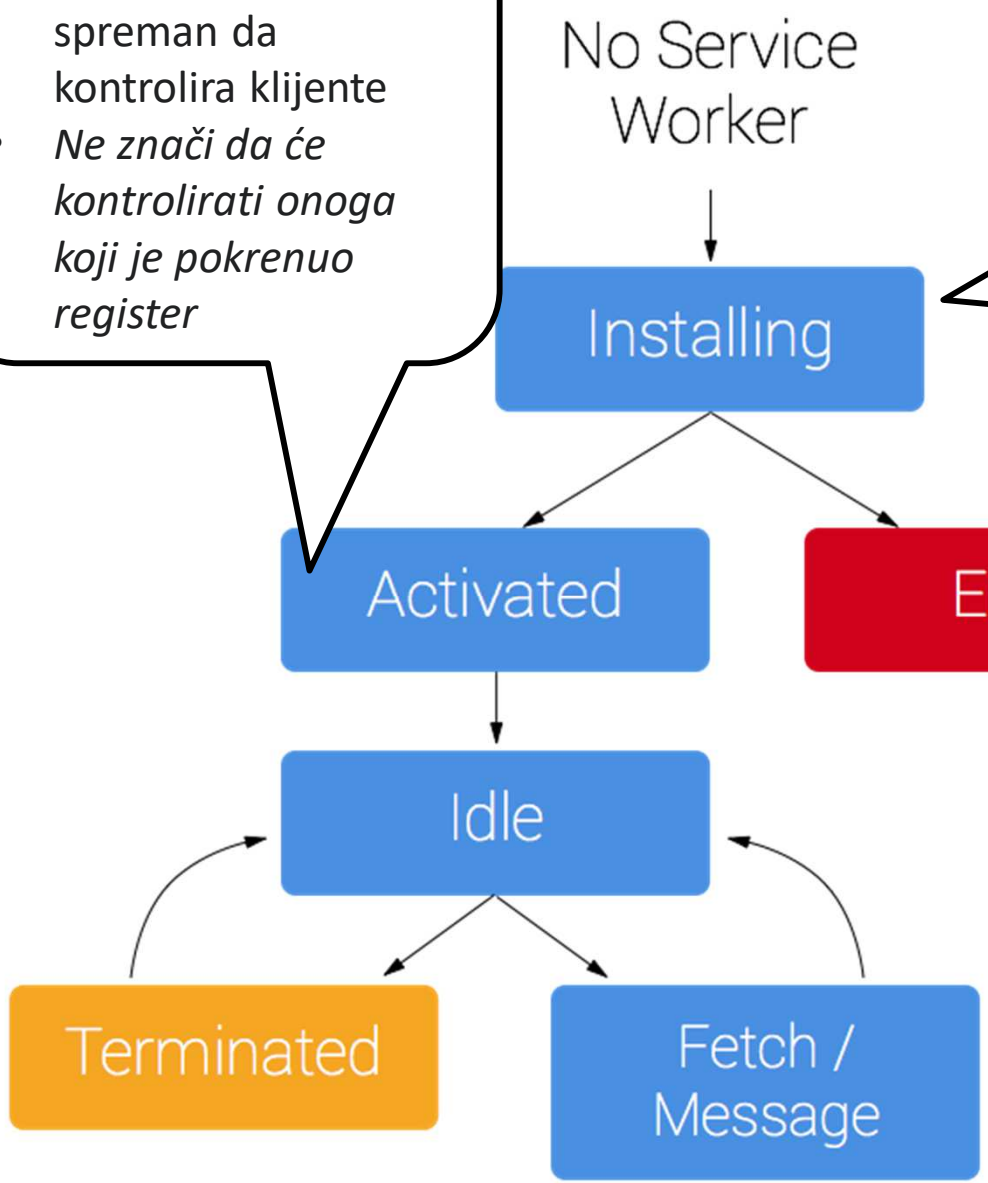


Životni ciklus SW-a

- SW moramo registrirati
  - `navigator.serviceWorker.register('/sw.js')`
  - Slijedi dohvat i instalacija (ako već nije instaliran)
- Po uspješnoj instalaciji i aktivaciji (vidi sljedeći slajd) SW će **kontrolirati** klijente (*clients*) u svom opsegu (***scope***)
  - „We call pages, workers, and shared workers **clients**"
  - **Kontrolirati** = mrežni zahtjevi idu preko SW-a; push notifikacije
  - Možemo provjeriti tko kontrolira s:
    `navigator.serviceWorker.controller` (null ili SW instanca)
  - Defaultni i maksimalni scope je **./** od URL-a skripte
    - `https://www.fer.unizg.hr/je/super/sw.js`
      `-> https://www.fer.unizg.hr/je/super`

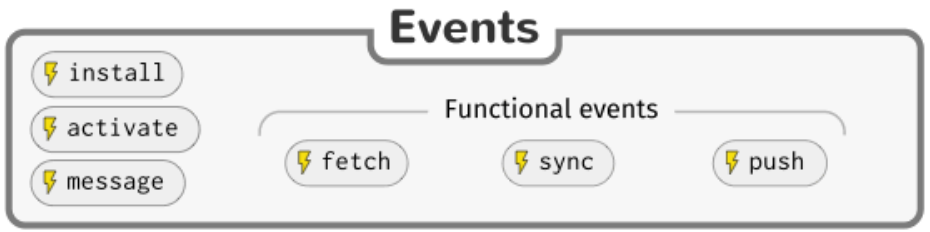
Životni ciklus *service workera*

- Aktivan ada je spreman da kontrolira klijente
- *Ne znači da će kontrolirati onoga koji je pokrenuo register*



- **Samo jednom**
- **Install event**
- S `installEvent.waitUntil()` možemo sačekati (promise) i provjeriti uspjeh instalacije
- Fetch i push idu mimo workera dok ne postane aktivan, ali ni tada ako nije stranica otvorena kroz SW
  - refresh
  - Ili `clients.claim()`

"document starts life with or without a Service worker and maintains that for its lifetime"



https://www.digitalocean.com/community/tutorials/demystifying-the-service-worker-lifecycle

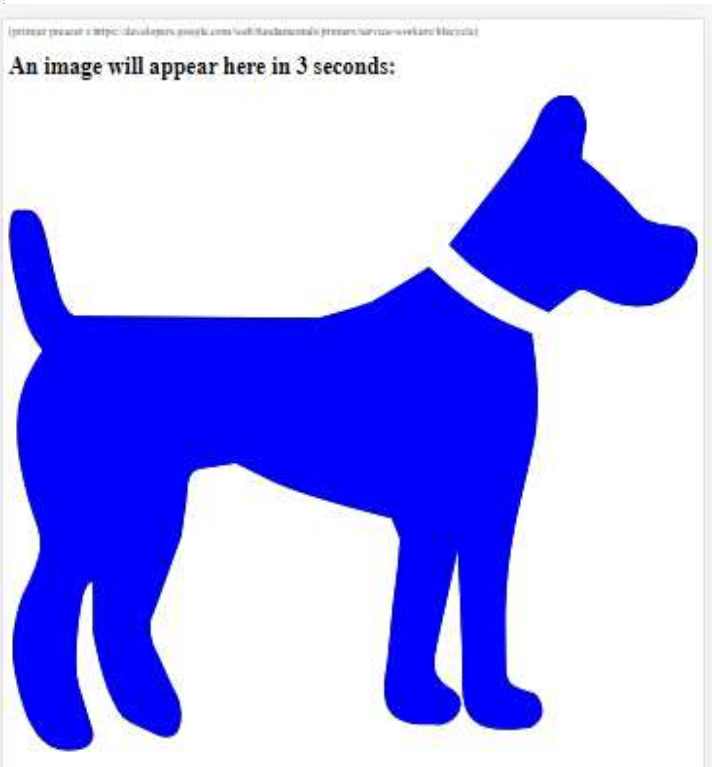
https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Cats and dogs – index.html

- Primjer preuzet s: https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle
- SW presreće zahtjev i umjesto psa vraća mačku (slj. slajd)
  - Ali: treba refresh da počne kontrolirati

```
<!DOCTYPE html>
<h1>An image will appear here in 3 seconds:</h1>
<script>
    navigator.serviceWorker.register('./sw.js')
        .then(reg => console.log('SW registered!', reg))
        .catch(err => console.log('Boo!', err));

    setTimeout(() => {
        const img = new Image();
        img.src = './dog.svg';
        document.body.appendChild(img);
    }, 3000);
</script>
```



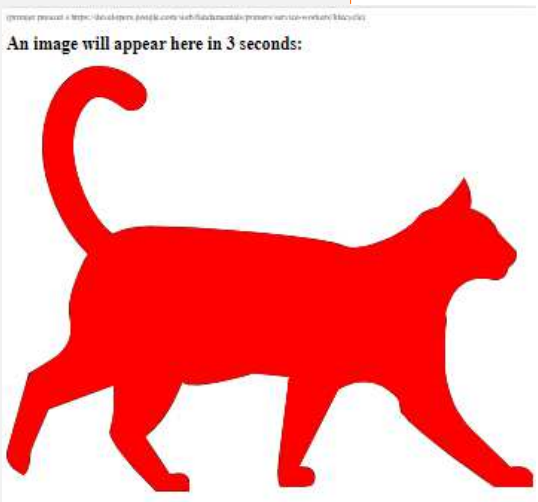

```
SW registered!                                                    (index):5
▸ ServiceWorkerRegistration {installing: ServiceWorker, waiting: null, active: null, navigationPreload: NavigationPreloadManager, scope: 'http
  s://service-worker-lifecycle-demos.glitch.me/main/', …}
V1 installing…                                                    sw.js:2
V1 now ready to handle fetches!                                   sw.js:10
> |
```

Cats and dogs – sw.js

- Puno novih stvari: cache API, self, SW events…

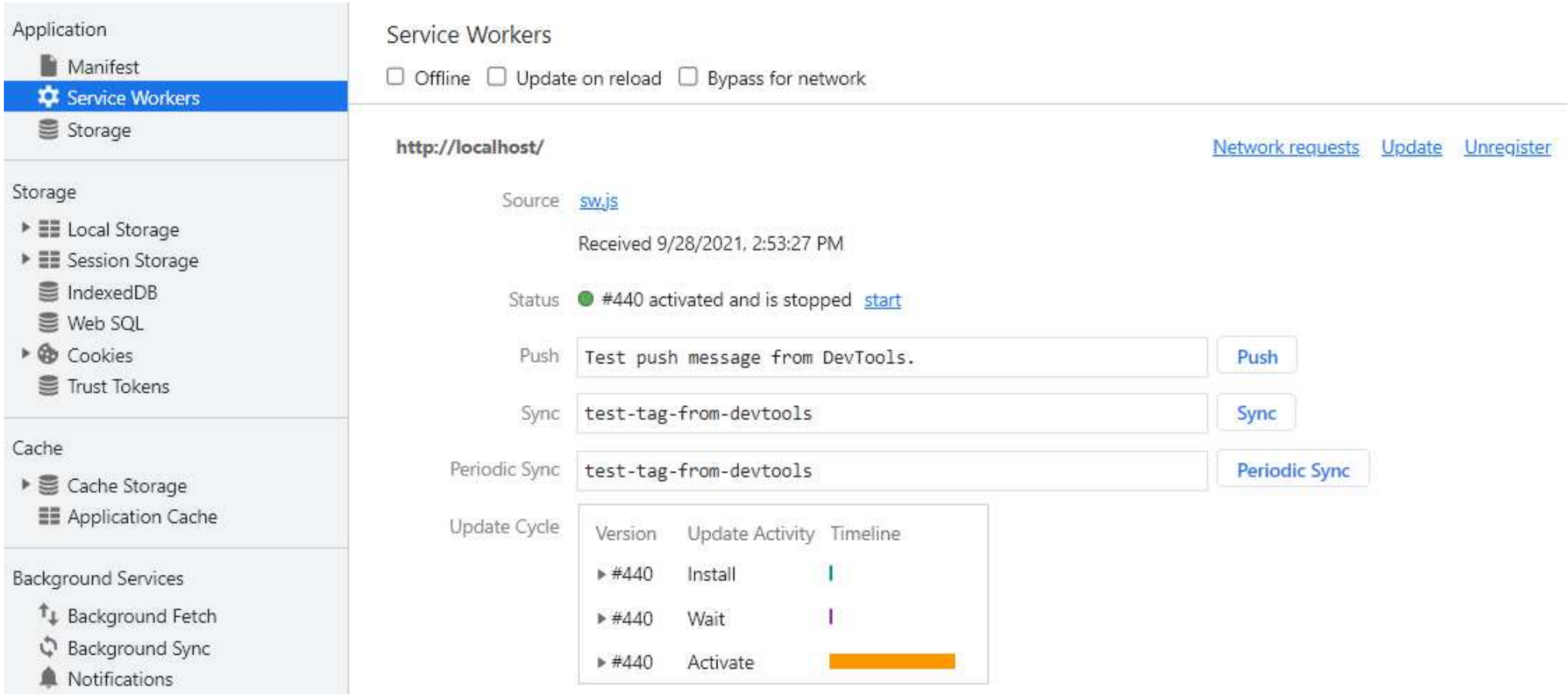
```
self.addEventListener('install', event => {
    console.log('V1 installing…');
    event.waitUntil( // cache a cat SVG
        caches.open('static-v1').then(cache => cache.add('./cat.svg'))
    );
});
self.addEventListener('activate', event => {
    console.log('Activated, V1 now ready to handle fetches!');
});
self.addEventListener('fetch', event => {
    const url = new URL(event.request.url);
    // serve the cat SVG from the cache if the request is
    // same-origin and the path is '/dog.svg'
    if (url.origin == location.origin && url.pathname == '/dog.svg') {
        event.respondWith(caches.match('/cat.svg'));
    }
});
```

https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage



Što keširati? CSS; JS; slike, itd,

Pogledajmo SW u devtools:

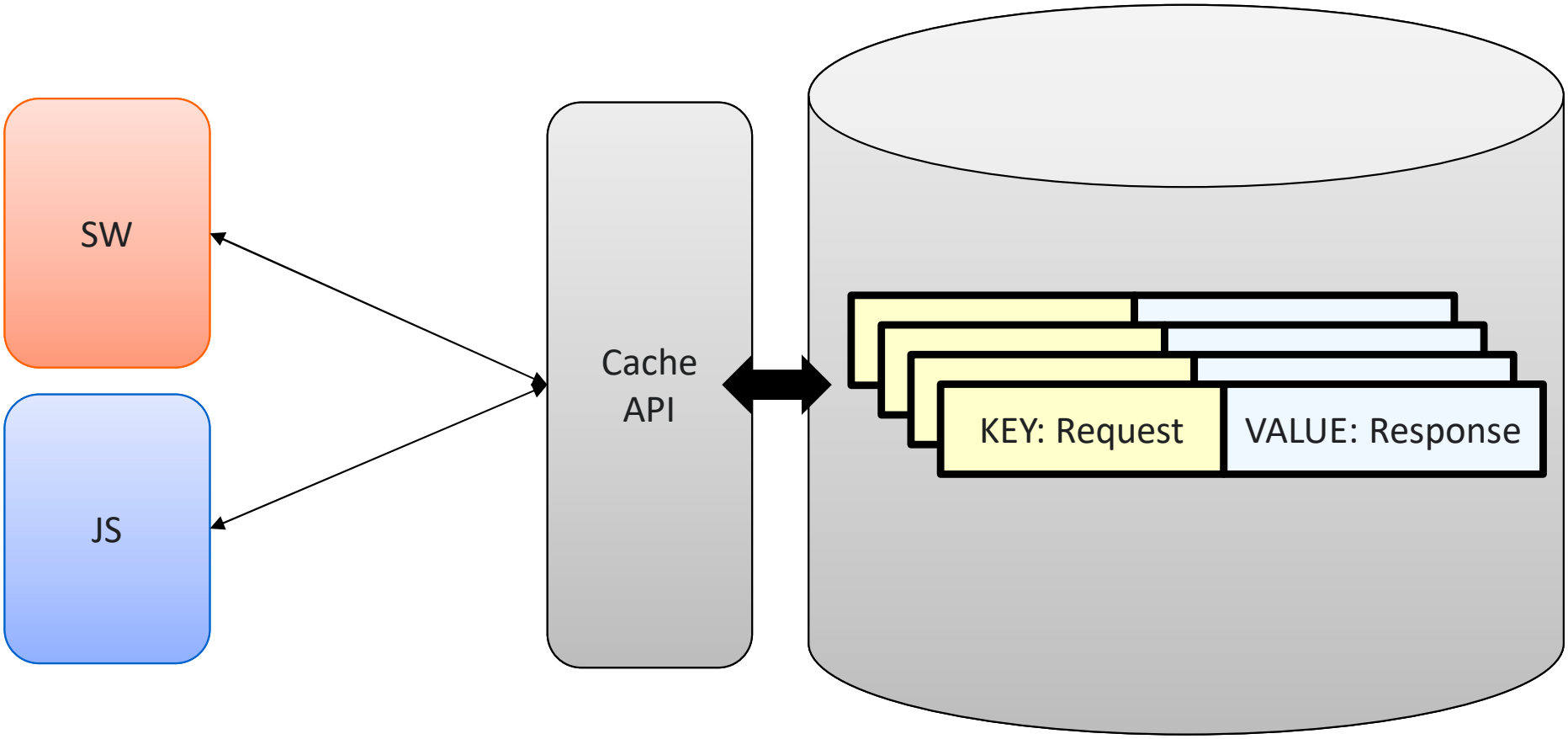


Ažuriranje SW-a

- Potrebno je promijeniti barem jedan bajt da se SW smatra novom verzijom
  - Mijenjanje URL-a SW-a nije dobra praksa
- ALI: novi SW će se instalirati ali neće postati aktivan dok barem jedna stranica (tab) koristi stari SW!
  - To znači da postoje dva SW-a: jedan (stari) aktivan i novi koji čeka
- Potrebno je:
  - Zatvoriti sve stranice/tabove da bi novi SW postao aktivan
  - Ili programski forsirati sa `self.skipWaiting()`
- U razvojnoj okolini puno lakše - **devtools**:
  - **update on reload**
  - Unregister/register
  - Skip waiting
- Detaljnije na: https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle#updates

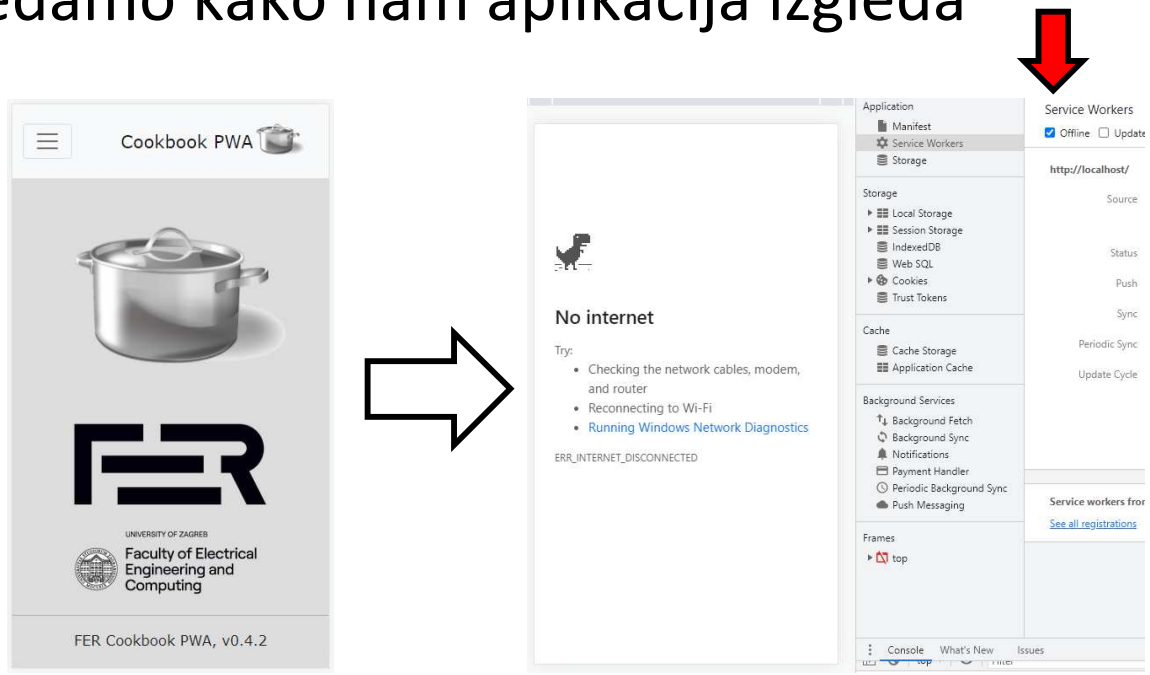
Cache API

- Key-value baza
- Može se koristiti i iz „običnog” JS-a



Što keširati?

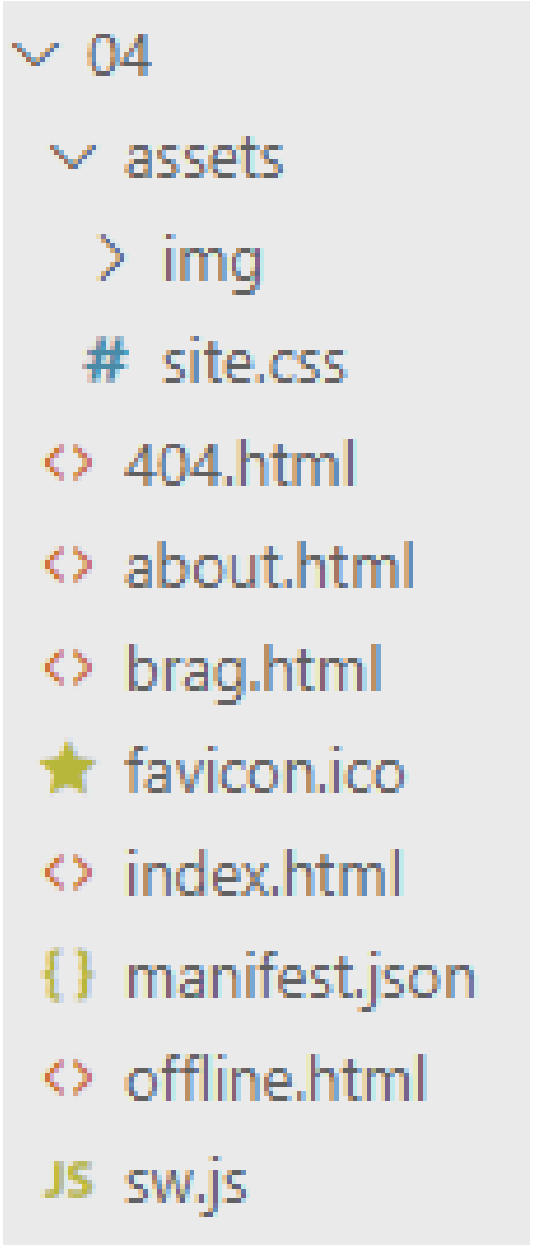
- Slike, CSS, JS, fontovi, …
- Primijetiti:
  - **Statički** sadržaj (keširati kod **instalacije**, ***app shell***)
  - **Dinamički** sadržaj
    - Keširati (ako ima smisla) za vrijeme rada aplikacije
- Keširanje kostura aplikacije (*app shell cacheing*):
  - Isključimo mrežu i pogledamo kako nam aplikacija izgleda
  - Popravimo, ponovimo



VER: 04

App shell + dinamičko keširanje

- Ovdje ćemo pokazati raskošnije rješenje koje:
  - **Djelomično** (popraviti – kako ih sve odrediti?) kešira app shell zahtjeve **kod instalacije**
  - **Dinamički kešira** sve ostale zahtjeve **kako zahtjevi dolaze**, te je u konačnici sve keširano
  - Uvodimo **404 stranicu** koju keširamo za slučaj 404
  - Uvodimo **Offline stranicu** koju možemo prikazati ako nismo keširali neku stranicu.



SW offline - instalacija

```
const filesToCache = [
    "/",
    "manifest.json",
    "index.html",
    "offline.html",
    "404.html",
  "https://fonts.googleapis.com/css2?family=Fira+Sans:ital,wght@0,400;0,700;1,400;1,700&display=swap",
  "https://fonts.gstatic.com/s/firasans/v11/va9E4kDNxMZdWfMOD5Vvl4jLazX3dA.woff2",
  "https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css",
];

const staticCacheName = "static-cache-v1";

self.addEventListener("install", (event) => {
    console.log("Attempting to install service worker and cache static assets");
    event.waitUntil(
        caches.open(staticCacheName).then((cache) => {
            return cache.addAll(filesToCache);
        })
    );
});
```

Popis **zahtjeva** (stringovi ne temelju kojih će se napraviti zahtjevi) koje ćemo inicijalno (kod **instalacije**) keširati

`event.waitUntil(PROMISE)`

Inače bi instalacija završila dok još nije keširano

Obavlja jedan po jedan request na zadani URL i kešira:
`(request, response)`

SW offline - fetch

VER: 04

```
self.addEventListener("fetch", (event) => {
  event.respondWith(
    caches
      .match(event.request)
      .then((response) => {
        if (response) {
          return response;
        }
        return fetch(event.request).then((response) => {
          if (response.status === 404) {
            return caches.match("404.html");
          }
          return caches.open(staticCacheName).then((cache) => {
            cache.put(event.request.url, response.clone());
            return response;
          });
        });
      })
      .catch((error) => {
        return caches.match("offline.html");
      })
  );
});
```

Pretražuje sve cacheve po ključu event.request

Pronašli u *cacheu*, vraćamo spremljeni odgovor

SW dohvaća s interneta

Potencijalno vraćamo spremljenu 404 stranicu

„Clone is needed because put() consumes the response body."

Prvo pohranjujemo odgovor u cache, te konačno vraćamo odgovor.
Sljedeći put bi morao biti u cacheu.
Je li ova strategija dobra - sve spremamo u isti cache i nakon nekog vremena više uopće ne idemo na internet?

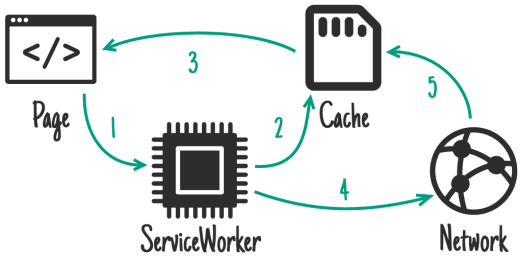
Bolji osnovni pristup

- Napraviti dva cachea (keys):
  - STATIC_CACHE_V<N>
  - DYNAMIC_CACHE
- Kod instalacije staviti app shell u statički cache
- Prilikom rada staviti datoteke (ne i JSON zahtjeve!) u dinamički cache
- Kod nove verzije SW-a, promijeniti verziju statičkog cachea, te:
  - Prilikom **aktivacije** obrisati sve druge (stare cacheve)
  - ✓ Dok novi SW nije aktiviran stari cachevi su još prisutni jer ih možda koriste druge aplikacije

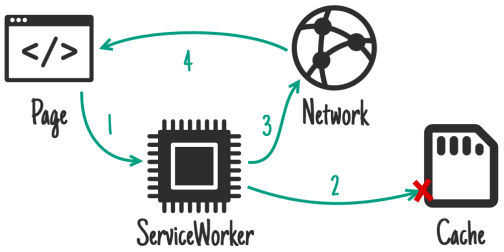
```
self.addEventListener("activate", (event) => {
  const cacheWhitelist = [staticCacheName];
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
            if (cacheWhitelist.indexOf(cacheName) === -1) {
                return caches.delete(cacheName);
            }
        })
      );
    })
  );
});
```

Strategije keširanja

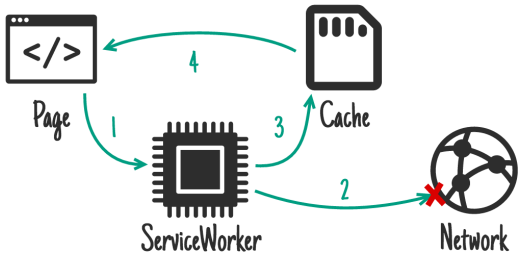
- Postoje brojne strategije keširanja, najpoznatije su:
  - ***Stale-While-Revalidate***



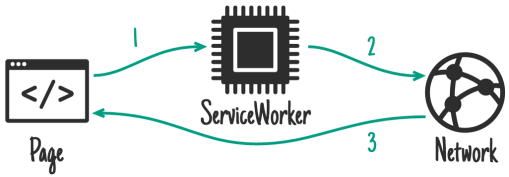
  - ***Cache First (Cache Falling Back to Network)***



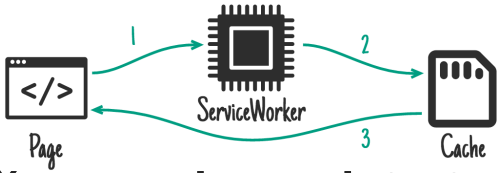
  - ***Network First (Network Falling Back to Cache)***



  - ***Network Only***



  - ***Cache Only***



- Primijetiti da možemo kombinirati strategije za različite skupine datoteka/zahtjeva

https://developers.google.com/web/tools/workbox/modules/workbox-strategies

Keširanje pomoću workboxa

Workbox

- Googleov alat koji uvelike pomaže razvoju SW-a
- Ugrađene strategije (s prethodnog slajda)
- Config -> generate -> sw.js
- Opaska: za cacheiranje podataka (JSON i sl.) bolje koristiti IndexedDB, a ne Cache Storage

```
import {ExpirationPlugin} from 'workbox-expiration';
import {registerRoute} from 'workbox-routing';
import {StaleWhileRevalidate} from 'workbox-strategies'; // Cache Google Fonts with a stale-while-revalidate strategy, with max num of entries.
registerRoute(
  ({url}) => url.origin === 'https://fonts.googleapis.com' ||
             url.origin === 'https://fonts.gstatic.com',
  new StaleWhileRevalidate({
    cacheName: 'google-fonts',
    plugins: [
      new ExpirationPlugin({maxEntries: 20}),
    ],
  }),
);
```

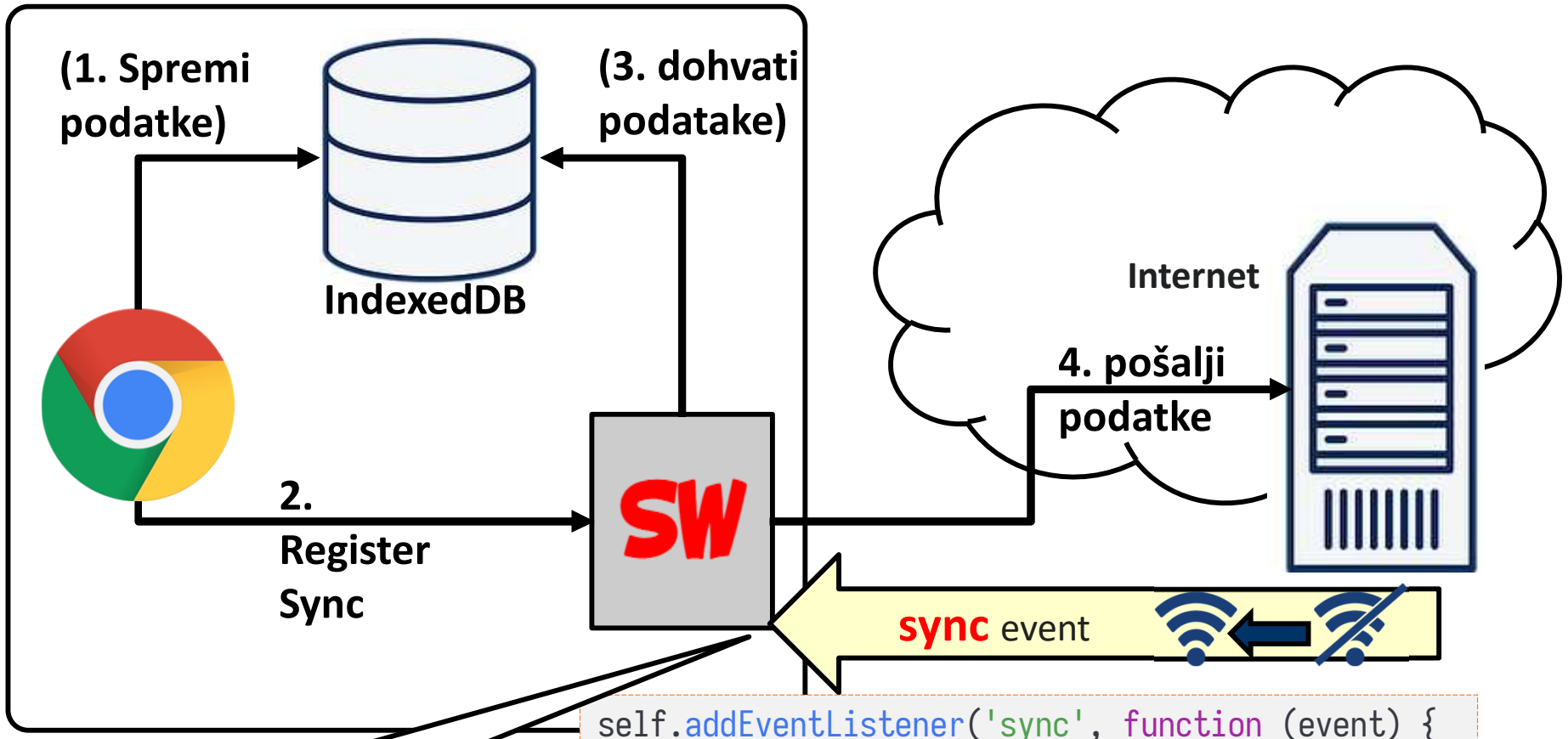
https://developers.google.com/web/tools/workbox

Pozadinska sinkronizacija

Background sync

Pozadinska sinkronizacija (*background sync*)

```
// Register sync - zatražimo jednokratnu sinkronizaciju
navigator.serviceWorker.ready.then(function(swRegistration) {
  return swRegistration.sync.register('myFirstSync');
});
```




```
self.addEventListener('sync', function (event) {
  if (event.tag == 'myFirstSync') {
    event.waitUntil(posaljiPodatke());
  }});
```

Radi i ako "zatvorimo tab"!

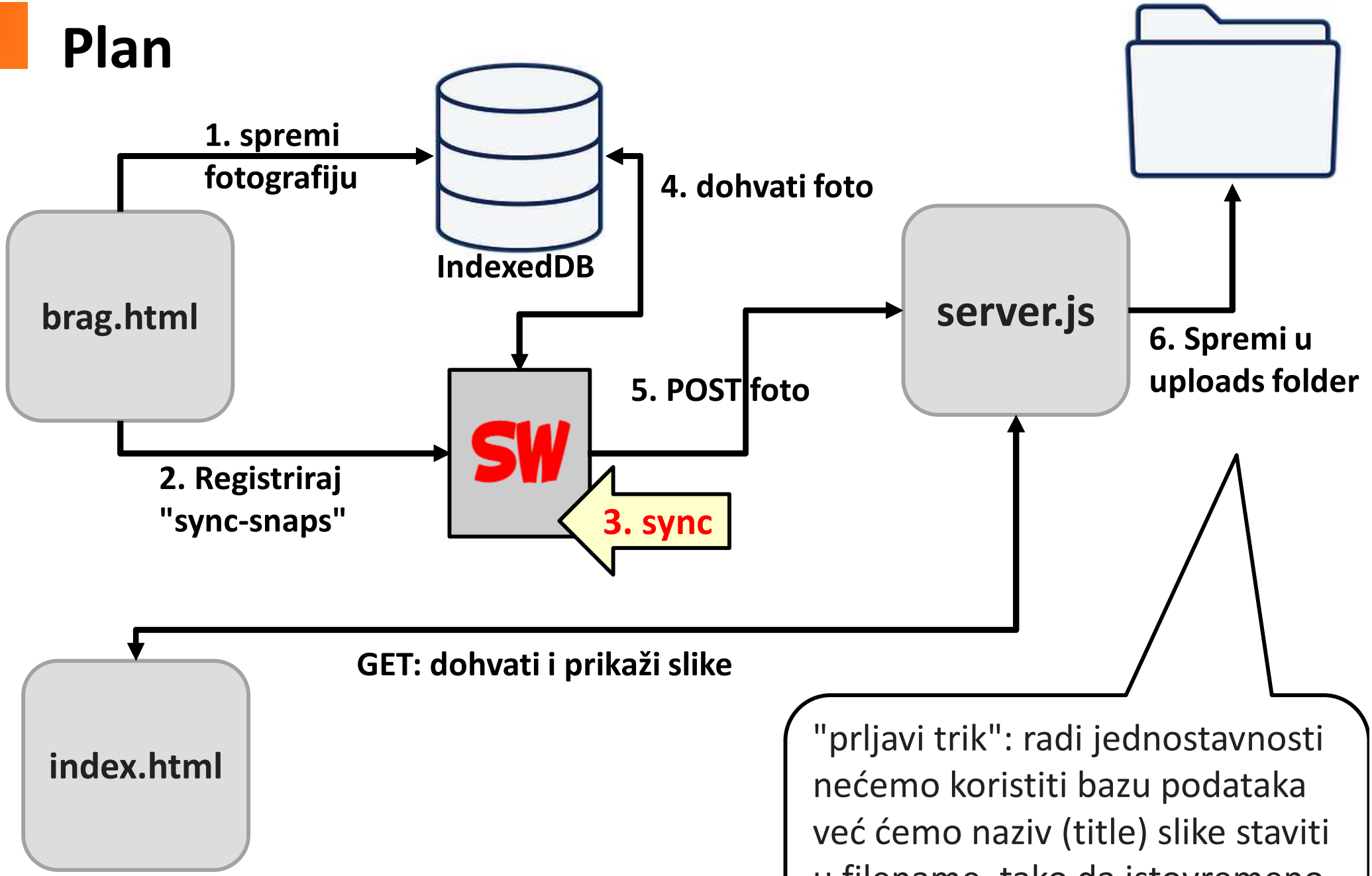
IndexedDB

API	Data Model	Persistence	Browser Support	Transactions	Sync/Async
File system	Byte stream	device	52%	No	Async
Local Storage	key/value	device	93%	No	Sync
Session Storage	key/value	session	93%	No	Sync
Cookies	structured	device	100%	No	Sync
Cache	key/value	device	60%	No	Async
IndexedDB	**hybrid**	**device**	**83%**	**Yes**	**Async**

https://blog.sessionstack.com/how-javascript-works-storage-engines-how-to-choose-the-proper-storage-api-da50879ef576

- Nažalost: API je prilično nezgrapan
- Nasreću:
  - https://github.com/jakearchibald/idb
  - https://www.npmjs.com/package/idb-keyval (trivijalni API)

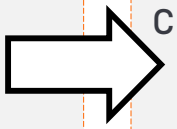
Plan


1. spremi fotografiju
IndexedDB
4. dohvati foto
brag.html
server.js
6. Spremi u uploads folder
5. POST foto
SW
2. Registriraj "sync-snaps"
3. sync
GET: dohvati i prikaži slike
index.html
"prljavi trik": radi jednostavnosti nećemo koristiti bazu podataka već ćemo naziv (title) slike staviti u filename, tako da istovremeno pohranimo i sliku i title

Kako fotografirati?

- Native API
  - Navigator.mediaDevices
  - Preko preglednika pristupamo kameri uređaja, bilo da je na pametnom telefonu, laptopu ili webcam na stolnom računalu
- "Trik": player -> canvas
  - Prikazan je ILI player ILI canvas
  - Konačno, iz canvasa generiramo png i pohranimo u IndexedDB

```
<video id="player" width="100%" autoplay>
</video>
navigator.mediaDevices
  .getUserMedia({ video: true,
                  audio: false })
  .then((stream) => {
     player.srcObject = stream;
  }).catch((err) => {...});
```



```
<canvas id="cnvFood"></canvas>
</video>
canvas
  .getContext("2d")
  .drawImage(player, 0, 0,
          canvas.width, canvas.height);
```

brag.html (1/2)

VER: 05

```
import { get, set } from "https://cdn.jsdelivr.net/npm/idb-keyval@6/+esm";
let player = document.getElementById("player");
let canvas = document.getElementById("cnvFood");
let beforeSnap = document.getElementById("beforeSnap");
let afterSnap = document.getElementById("afterSnap");
let snapName = document.getElementById("snapName");
let startCapture = function () {
    beforeSnap.classList.remove("d-none");
    afterSnap.classList.add("d-none");
    if (!("mediaDevices" in navigator)) {
        // fallback to file upload button, ili sl.
        // vidjet i custom API-je: webkitGetUserMedia i mozGetU
    } else {
        navigator.mediaDevices
            .getUserMedia({ video: true, audio: false })
            .then((stream) => {
                player.srcObject = stream;
            })
            .catch((err) => {
                alert("Media stream not working");
                console.log(err);
            });
    }
};
startCapture();
```

Sakrijemo canvas, pokažemo player

U PE/GD smislu bi trebalo obraditi preglednike koji ne podržavaju mediaDevices i omogućit button za file upload

Video stream pridružujemo video elementu

Pokrećemo, čim se otvori stranica brag.html

brag.html (2/2)

VER: 05

```
document.getElementById("btnSnap").addEventListener("click", function (event) {
    canvas.width = player.getBoundingClientRect().width;
    canvas.height = player.getBoundingClientRect().height;
    canvas.getContext("2d")
        .drawImage(player, 0, 0, canvas.width, canvas.height);
    stopCapture();
});
document.getElementById("btnUpload") addEventListener("click", function (event) {
    ...
    if ("serviceWorker" in navigator && "SyncManager" in window) {
        fetch(canvas.toDataURL())
            .then((res) => res.blob())
            .then((blob) => {
                let ts = new Date().toISOString();
                let id = ts + snapName.value.replace(/\s/g, "_"); // ws->_
                set(id, { id, ts, title: snapName.value, image: blob });
                return navigator.serviceWorker.ready;
            }).then((swRegistration) => {
                return swRegistration.sync.register("sync-snaps");
            }).then(() => {
                console.log("Queued for sync");
                startCapture();
            }).catch((err) => { console.log(error); });
    } else { // fallback: pokusati poslati, pa ako ima mreze onda dobro...
        alert("TODO - vaš preglednik ne podržava bckg sync...");
    }
});
```

Kopiramo sadržaj playera u canvas i zaustavljamo stream i skrivamo player tj. otkrivamo canvas

Pretvaramo Base64 kodiranu sliku s canvasa u blob, te ju pohranjujemo u IndexedDB

Registriramo jednokratni sync event. Ako je mreža dostupna, sync event nad SW-om će biti odmah okinut.

VER: 05

sw.js

```
self.addEventListener("sync", function (event) {
  if (event.tag === "sync-snaps") {
    event.waitUntil(syncSnaps());
  }
});
let syncSnaps = async function () {
  entries().then((entries) => {
    entries.forEach((entry) => {
      let snap = entry[1]; //  Each entry is an array of [key, value].
      let formData = new FormData();
      formData.append("id", snap.id);
      formData.append("ts", snap.ts);
      formData.append("title", snap.title);
      formData.append("image", snap.image, snap.id + ".png");
      fetch("/saveSnap", {
        method: "POST",
        body: formData,
      }).then(function (res) {
          if (res.ok) {
              res.json().then(function (data) {
                  console.log("Deleting from idb:", data.id);
                  del(data.id);
              });
          } else {console.log(res);}
      }).catch(function (error) {console.log(error);});
    });
  });
};
```

Obrađujemo naš problem

idb-keyval na ovaj način dohvaća sve zapise iz IndexedDB baze

Sklapamo i šaljemo POST zahtjev koji uključuje i našu sliku. Podsjetnik: koji je content-type ovog zahtjeva?

Ako smo uspjeli server.js će nam vratiti nazad id koji smo poslali koji sad koristimo da obrišemo zapis iz IndexedDB

ZOKŽZV: Periodička pozadinska sinkronizacija

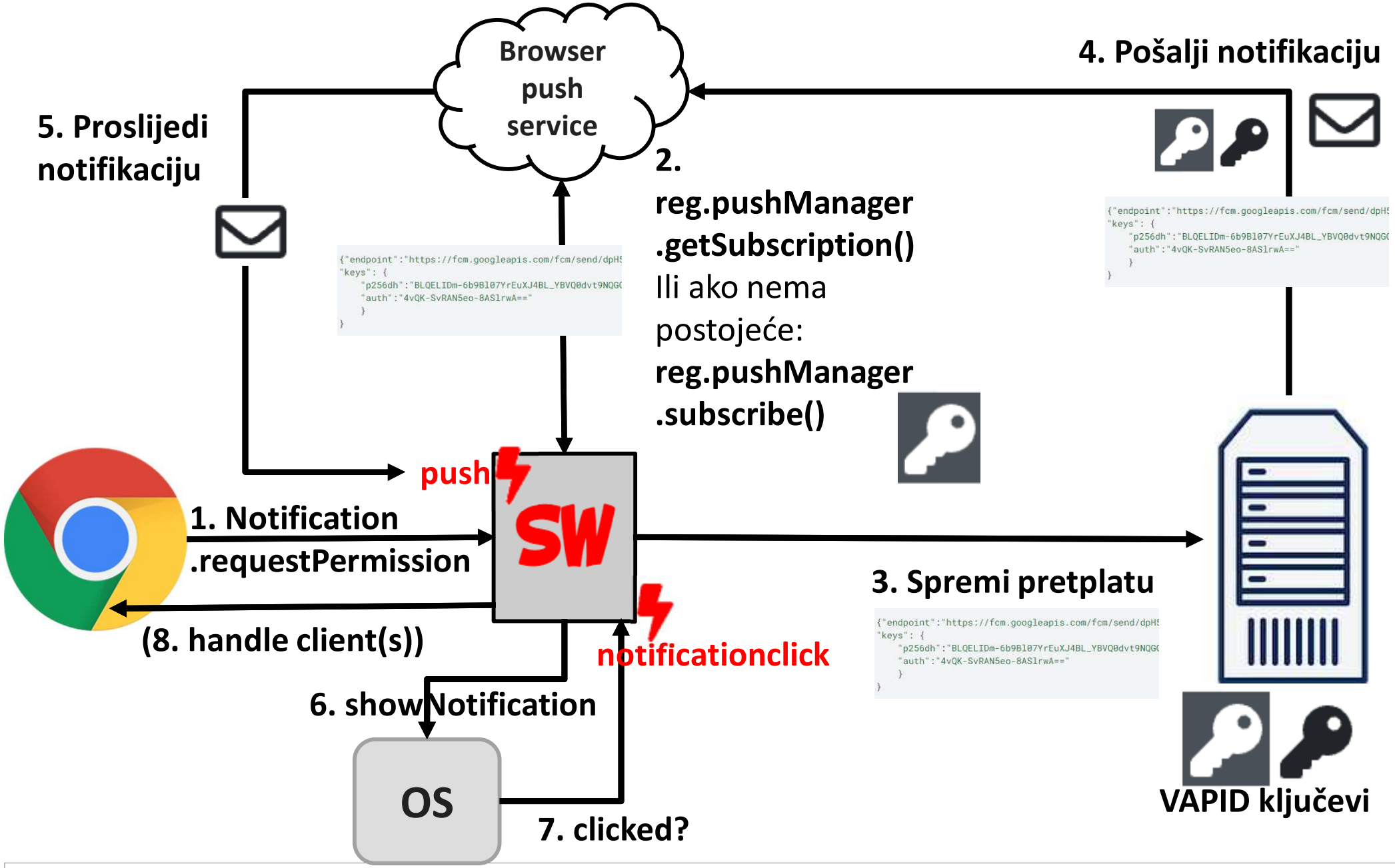
- Nalik "običnoj" ali se periodički ponavlja u zadanom intervalu, npr: https://developer.mozilla.org/en-US/docs/Web/API/Web_Periodic_Background_Synchronization_API

```
async function registerPeriodicNewsCheck() {
  const registration = await navigator.serviceWorker.ready;
  try {
    await registration.periodicSync.register('get-latest-news', {
      minInterval: 24 * 60 * 60 * 1000,
    });
  } catch { console.log('Periodic Sync could not be registered!'); }
}
```

```
self.addEventListener('periodicsync', event => {
  if (event.tag == 'get-latest-news') {
    event.waitUntil(fetchAndCacheLatestNews());
  }
});
```

Push notifikacije

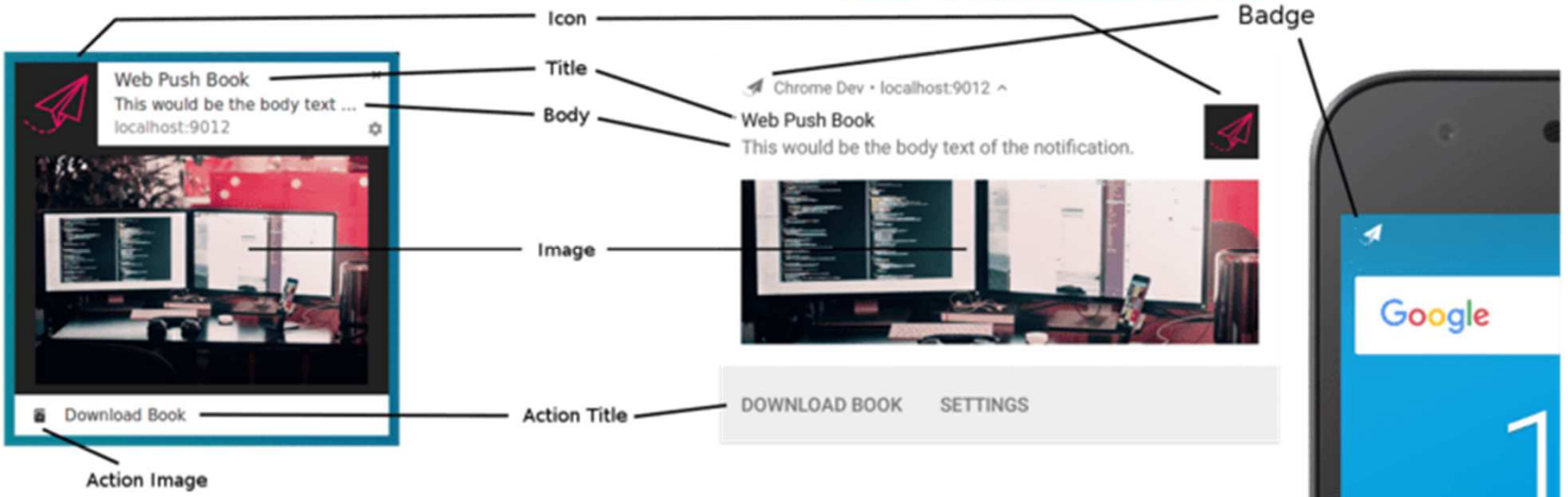
Kako rade push notifikacije?



Primijetiti – notifikacije su nezavisne od SW-a…

- … iako se najčešće koriste u kombinaciji sa SW-om
  - Kada tražimo dozvolu za notifikaciju, odmah dobijemo i implicitnu dozvolu za push
- Ali, npr. možemo koristiti i bez SW-a, npr.:
  - Google photos – pokrenemo backup i onda aplikacija radi backup u pozadini te nas obavijesti putem notifikacije kad je gotova
- Opcije:

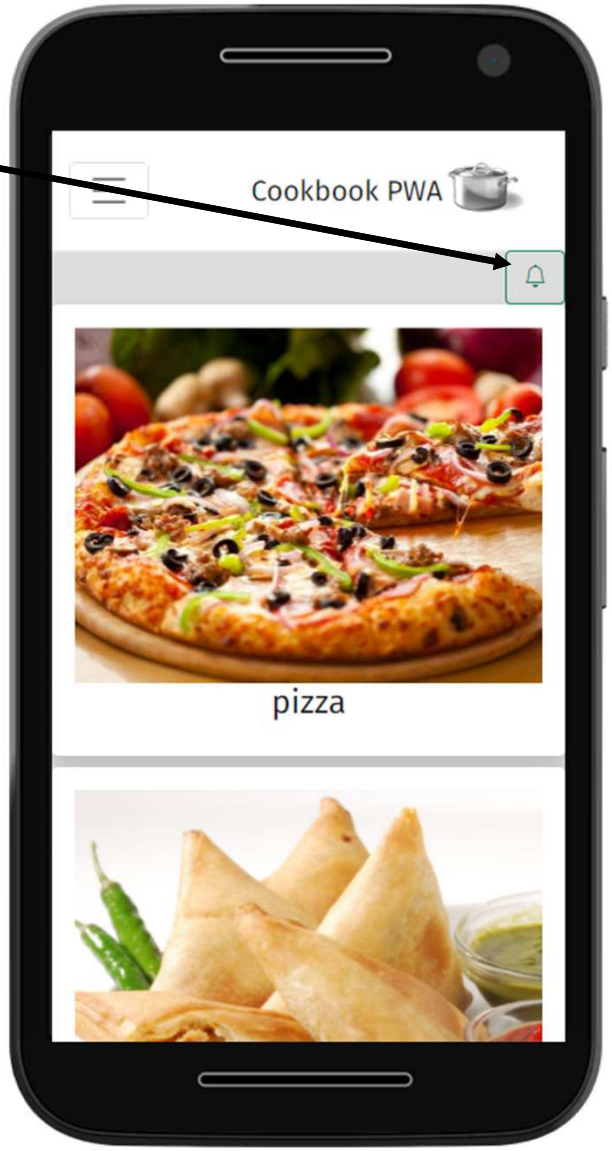
https://developers.google.com/web/fundamentals/push-notifications/display-a-notification



- Demo: https://web-push-book.gauntface.com/demos/notification-examples/

Plan

- Dodati gumb u index.html gdje se može pretplatiti na notifikacije
  - Istovremeno i push dozvola
- Spremiti pretplatu na server
  - Array (in memory)
  - Pohranjujemo u datoteku
- Kada se snimi slika poslati notifikaciju
- Prikazati notifikaciju u SW-u
  - OS prikazuje notifikaciju
  - Kada korisnik klikne, SW event
- Preusmjeriti klijente koje kontrolira SW na stranicu koja je zadana u notifikaciji (index.html)



https://floating-oasis-41640.herokuapp.com/

VAPID ključevi, web-push

- **npm i web-push**
- web-push generate-vapid-keys
- Ili preko npm run:

```
λ cat package.json | grep gen
    "gen-vapid": "web-push generate-vapid-keys"

C:\Users\Igor\OneDrive - fer.hr\Nastava\Web2 - Napredni razvoj programske potpore za web\wip-p
λ npm run gen-vapid

> pwa-examples@1.0.0 gen-vapid
> web-push generate-vapid-keys


=======================================

Public Key:
BBfae6kt70vtdHKE_w3sd2c9viue80_wUXE6ZMjkRprWCtjQ5ZgzqGDWxkc79ncxc2LCSGuFnVNYrJSYX9NWG8Y

Private Key:
fIgVNhAHkYrhJXNWca_4qTaykZamA4Y1xdDGrXVAtxc

=======================================


C:\Users\Igor\OneDrive - fer.hr\Nastava\Web2 - Napredni razvoj programske potpore za web\wip-p
λ npm run gen-vapid

> pwa-examples@1.0.0 gen-vapid
> web-push generate-vapid-keys


=======================================

Public Key:
BMpYzuKrXO0AEP8h9dagpxkz6f4BLgbWjToFz8t228MWYlpccGHLxU8LM7f1y2X6sImk8aIBID0v-RVDqTwNmNI

Private Key:
AenZ17i7lAD8Mdi6QujHEEEpEQU1axXYF3dTgxb2KiM

=======================================
```

- Trebat će nam:
  - Na klijentu:
    - Public key
  - Na serveru:
    - Public key
    - Private key

push.js (iz index.html)

VER: 06

```
if ("Notification" in window
  && "serviceWorker" in navigator) {
  btnNotif.addEventListener("click", function () {
    Notification.requestPermission(
     async function (res) {
      if (res === "granted") {
        await setupPushSubscription();
      } else {
        console.log("User denied push notifs:", res)
      }
    });
  });
} else {
  btnNotif.setAttribute("disabled", "");
  btnNotif.classList.add("btn-outline-danger");
}
```

```
async function setupPushSubscription() {
  try {
    let reg = await navigator.serviceWorker.ready;
    let sub = await reg.pushManager.getSubscription();
    if (sub === null) {
      let publicKey = "BL1oXiSXCjK(...)dT2EJGH33qe5iw";
      sub = await reg.pushManager.subscribe({
          userVisibleOnly: true,
          applicationServerKey: urlBase64ToUint8Array(publicKey)
      })
      let res = await fetch("/saveSubscription", {
        method: "POST", headers: {
        "Content-Type": "application/json",
        Accept: "application/json",
       }, body: JSON.stringify({ sub }),
      });
      if (res.ok) {
        alert("Yay, subscription generated a   saved:\n" +
           JSON.stringify(sub));
      }
    } else { alert("You are alreay subscribed
    } catch (error) {
      console.log(error);
    }
}
```

Javni VAPID ključ

Preglednik pita korisnika za dozvolu

Kontraktiramo preglednikov push service, ključ ne može ići plain-text, moramo ga prekodirati

Pohranjujemo pretplatu „kod sebe”

server.js (pohrana pretplata i slanje notifikacija)

VER: 06

```
const webpush = require('web-push');
// Umjesto baze podataka, čuvam pretplate u
// datoteci:
let subscriptions = [];
const SUBS_FILENAME = 'subscriptions.json';
try {
  subscriptions =
JSON.parse(fs.readFileSync(SUBS_FILENAME));
} catch (error) {
  console.error(error);
}

app.post("/saveSubscription", function(req, res) {
  let sub = req.body.sub;
  subscriptions.push(sub);
  fs.writeFileSync(SUBS_FILENAME,
     JSON.stringify(subscriptions));
  res.json({
    success: true
  });
});
```

Poziva se iz već viđene saveSnaps

```
async function sendPushNotifications(snapTitle) {
  webpush.setVapidDetails('mailto:ime.prezime@fer.hr',
  'BL1oXi(...)T2EJGH33qe5iw',
  '4B9u(...)BZshEZnI');
  subscriptions.forEach(async sub => {
    try {
      await webpush.sendNotification(sub, JSON.stringify({
        title: 'New snap!',
        body: 'Somebody just snaped a new photo: ‘
              + snapTitle,
        redirectUrl: '/index.html'
       }));
    } catch (error) {
      console.error(error);
    }
  });
}
```

payload

sw.js (push notificationclick)

VER: 06

```
self.addEventListener("push", function (event) {
  var data = { title: "title", body: "body",
               redirectUrl: "/" };

  if (event.data) {
    data = JSON.parse(event.data.text());
  }
  let options = {
    body: data.body,
    icon: "assets/img/android/android-
launchericon-96-96.png",
    badge: "assets/img/android/android-
launchericon-96-96.png",
    vibrate: [200, 100, 200, 100, 200, 100, 200],
    data: {
      redirectUrl: data.redirectUrl,
    },
  };

  event.waitUntil(
    self.registration.showNotification(
      data.title, options)
  );
});
```

Na neki način, sami sebi šaljemo, u drugi event od SW-a

```
self.addEventListener("notificati      k", function (event) {
  let notification = event.noti      tion;
  // mogli smo i definirati       ons, pa ovdje granati s
  //obzirom na:  event.ac      on;
  event.waitUntil(
    clients.matchAll().then(function (clis) {
      clis.forEach((client) => {
        client.navigate(notification.data.redirectUrl);
        client.focus();
      });
      notification.close();
    })
  );
});

self.addEventListener("notificationclose", function (event) {
  console.log("notificationclose", event);
});
```

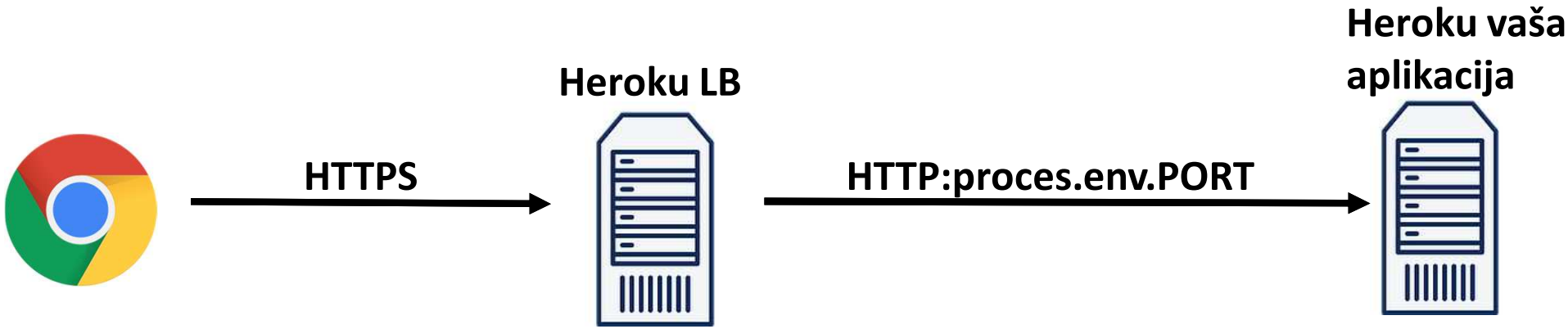
Vraća sve klijente koje kontrolira ovaj SW

Otvaramo "/index.html" koji smo zadali još u server.js

Postoji i `notificationclose`, ali ga ne koristimo u ovom scenariju

Nekoliko tehničkih opaski

- Dominantno razvijamo na:
  - Chrome
  - http://localhost (jer ako je localhost onda ne mora biti https)
  - Samopotpisani certifikati i lokalni HTTPS server nam nisu od pomoći:
    - Neće raditi ni manifest install
    - Neće registrirati SW
- Kako onda testirati s mobilnog uređaja?
  - Free account na https://www.heroku.com/free
  - https://devcenter.heroku.com/articles/getting-started-with-nodejs
  - Dovoljno je imati http server:




```
const httpPort = process.env.PORT || 5000;
```

Korisni izvori

- https://developers.google.com/web/ilt/pwa
- https://developers.google.com/web/tools/workbox
- https://github.com/hemanth/awesome-pwa
- https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
- https://serviceworke.rs/