Laboratorium 1

Progresywna Web Aplikacja (**PWA**)

Celem laboratorium jest utworzenie strony internetowej spełniającej wymagania PWA.

Aplikacja **PWA** jest zbiorem dobrych praktyk, których przestrzeganie spowoduje, że aplikacja napisana w językach HTML, CSS, JS będzie dostosowywała się do urządzenia użytkownika i wykorzystywała jego peryferia symulując tym samym aplikację natywną np. wykorzystanie kamery i mikrofonu, push notyfikacje, możliwość pracy offline, itp.

Aplikacje PWA muszą spełnić 3 wymogi:

1. Hostowanie aplikacji protokołem HTTPS
2. Dołączenie Service Workera
3. Plik konfigurujący (manifest.json)

Service Worker jest plikiem JavaScript, który pełni rolę pośrednika (proxy) pomiędzy aplikacją a siecią. Dzięki temu istnieje możliwość zapisywania (cache'owania) zawartości aplikacji (dostęp offline) jak również przyśpieszanie renderowania. Service Worker działa w tle aplikacji i jest niezależny od głównej logiki oraz nie wymaga żadnych elementów UI. Service Worker nie jest dostępny podczas pierwszego odwiedzenia strony internetowej. Wchodząc po raz pierwszy na określoną stronę w technologi PWA, nie zobaczymy od razu okna z propozycją instalacji. Dzieje się tak, ponieważ podczas pierwszego uruchomienia strony Service Worker musi się zainstalować i dopiero kolejne przerenderowanie strony odpali nam Service Worker'a.

Manifest.json jest to plik w którym umieszczone są najważniejsze informacje na temat aplikacji. Informacje te są wykorzystywane przez urządzenie użytkownika w celu poprawnego renderowania aplikacji na ekranie. Plik ten zawiera informacje takie jak:

- nazwa pełna i skrócona aplikacji
- lokalizacja ikon
- początkowy adres URL (relatywnie do adresu domeny)
- domyślne ustawienie pozycji ekaranu
- splash screen (ekran widoczny podczas startowania aplikacji)

Zadania do wykonania:

1. Utworzenie strony internetowej (hmtl+css+js)
2. Instalacja wtyczki Lighthouse (Google Chrome) - https://developers.google.com/web/tools/lighthouse/
3. Stworzenie pliku /manifest.json

```
{
    "short_name": "",
    "name": "",
    "icons": [
        {
```

```
            "src": "",
            "sizes": "144x144",
            "type": "image/png"
        },
        {
            "src": "",
            "sizes": "192x192",
            "type": "image/png"
        }
    ],
    "start_url": "/index.html",
    "background_color": "",
    "theme_color": "",
    "display": "fullscreen"
}
```

4. Połączenie dokumentu HTML z manifestem:

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

5. Stworzenie pliku /service-worker.js

```
const CACHE_NAME = 'nazwa';

// List of files which are store in cache.
let filesToCache = [
    '/',
    '/css/style.css',
    '/images/logo.jpg,
    '/js/main.js'
];

self.addEventListener('install', function (evt) {
    evt.waitUntil(
        caches.open(CACHE_NAME).then(function (cache) {
            return cache.addAll(filesToCache);
        }).catch(function (err) {
            //console.error(err);
        })
    );
});

self.addEventListener('fetch', function (evt) {
    // console.log(event.request.url);
    evt.respondWith(
        // Firstly, send request..
        fetch(evt.request).catch(function () {
            // When request failed, return file from cache...
            return caches.match(evt.request);
        })
    );
});
```

6. Dołączenie service workera

```
<script>
const PATH = 'service-worker.js';
```

```
let isServiceWorkersSupport = ('serviceWorker' in navigator);

if (isServiceWorkersSupport) {
    console.log('Will service worker register?');
    navigator.serviceWorker.register(PATH).then(function () {
        console.log("Yes it did.");
    }).catch(function (err) {
        console.log("No it didn't. This happened: ", err)
    });
}
</script>
```

7. Umieszczenie plików aplikacji na serwerze www
8. Sprawdzenie aplikacji w Lighthouse i poprawa błędów