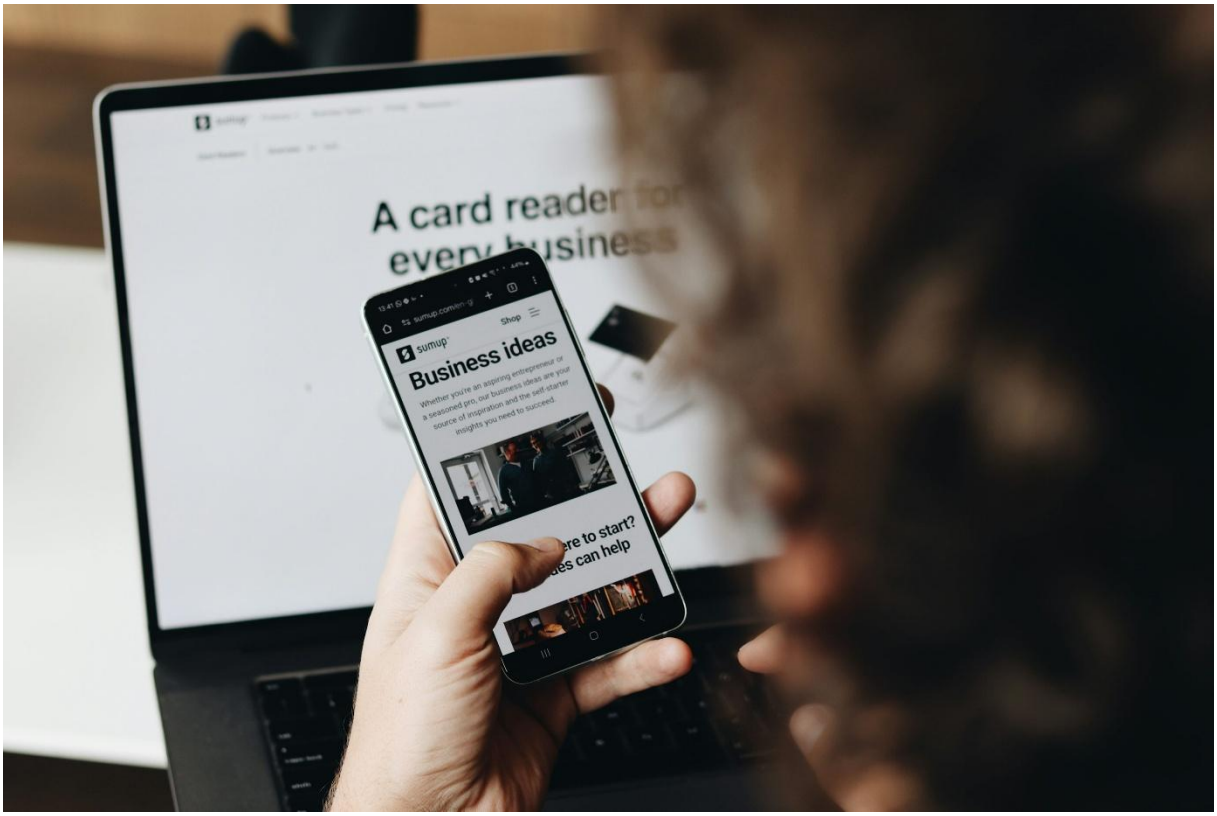
Programowanie w chmurze Laboratorium 3. Aplikacje chmurowe typu PWA.



Wprowadzenie

Progressive Web App to aplikacja internetowa uruchamiana tak jak zwykła strona internetowa, ale umożliwiająca stworzenie wrażenia działania jak natywna aplikacja mobilna (lub aplikacja desktopowa). Więcej : https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/

Aby stworzyć i uruchomić Progressive Web App (PWA), należy spełnić kilka kluczowych wymagań technicznych oraz wybrać odpowiednie środowisko do hostowania aplikacji. Oto szczegółowe informacje i wymagania dla PWA:

Aplikacje PWA muszą być serwowane przez bezpieczne połączenie HTTPS. Jest to wymagane, aby zapewnić prywatność użytkowników, bezpieczeństwo danych oraz autentyczność treści. Wyjątkiem jest lokalne środowisko deweloperskie, gdzie aplikacje mogą działać na adresach localhost lub 127.0.0.1 bez HTTPS. Wymagany jest Service Worker czyli skrypt JavaScript, który działa w tle i umożliwia obsługę funkcji takich jak cache, tryb offline czy powiadomienia push. Jest kluczowy dla działania PWA, ponieważ pozwala na przechwytywanie żądań sieciowych i zarządzanie pamięcią podręczną.

Plik manifest.json definiuje podstawowe informacje o aplikacji, takie jak nazwa, ikony, kolory motywu oraz sposób wyświetlania (np. pełnoekranowy tryb „standalone”). Manifest umożliwia instalację aplikacji na urządzeniach użytkowników. Aplikacja PWA musi być responsywna i działać na różnych urządzeniach oraz przeglądarkach. Powinna oferować podstawową funkcjonalność na starszych przeglądarkach, a bardziej zaawansowane funkcje na nowoczesnych. Należy pamiętać, iż optymalizacja zasobów, takich jak obrazy (np. format WebP), minifikacja kodu CSS/JS oraz szybkie ładowanie aplikacji są kluczowe dla wydajności.

Tworzymy aplikację PWA z wykorzystaniem HTML, CSS oraz JS.

Spróbujmy utworzyć pierwszą naszą aplikację typu PWA wykorzystując VanillaJS i środowisko Visual Studio Code. Utworzymy plik manifestu oraz Service Worker aby stworzyć prawie najprostszą możliwą aplikację PWA, taką, która działa bez połączenia z Internetem i może być dodana do ekranu głównego urządzenia mobilnego. Aplikacja powinna wyświetlać niektóre treści, nawet jeśli JavaScript jest wyłączony. Zapobiega to wyświetlaniu pustej strony przez użytkowników, jeśli ich połączenie internetowe jest słabe lub jeśli używają starszej przeglądarki.

1. Utwórz nowy projekt o nazwie PWA-JS w wybranym przez siebie środowisku do tworzenia aplikacji internetowych. Utwórz katalog js w którym znajdzie się implementacja kodu JavaScript. Struktura katalogów powinna wyglądać następująco:

   ```
   PWA-JS/
   │
   ├── index.html
   ├── style.css
   ├── manifest.json
   ├── sw.js
   ├── js/
   │   └── main.js
   └── images/
       └── (ikony aplikacji)
   ```

2. Utwórz plik o nazwie index.html w folderze głównym projektu, który powinien wyglądać mniej więcej tak:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>PWA PIAC TEST</title>
  <link rel="manifest" href="/manifest.json">
  <link rel="stylesheet" href="style.css">
  <link rel="icon" href="favicon.ico" type="image/x-icon">
  <link rel="apple-touch-icon" href="images/pwa-icon-152.png">
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar" content="#db4938">
  <meta name="theme-color" content="#db4938">
  <meta name="apple-mobile-web-app-title" content="PWA PIAC">
</head>
<body>
  <div class="container">
    <main>
      <h1 class="title">Hello World!</h1>
      <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry...</p>
    </main>
```

```
  </div>
  <script src="js/main.js"></script>
</body>
</html>
```

3. Następnie utwórz plik o nazwie style.css i dodaj ten kod:

```
html, body {
   font-family: sans-serif;
   text-align: center;
   height: 100%;
   margin: 0;
   padding: 0;
   width: 100%;
}

.container {
   margin: auto;
   text-align: center;
}

main {
   padding: 2rem;
   font-size: 1rem;
}

.title {
   font-size: 3rem;
}
```

4. Następnym zadaniem jest utworzenie i dodanie Service Worker’a. Service Worker może przechwytywać i obsługiwać żądania sieciowe, zarządzać pamięcią podręczną, aby włączyć obsługę offline lub wysyłać powiadomienia push do użytkowników. W przypadku naszej aplikacji użyjemy jednego do pobrania i buforowania naszej zawartości, a następnie udostępnienia jej z powrotem z pamięci podręcznej, gdy użytkownik jest offline.
5. Tworzymy teraz Service Worker do obsługi pamięci podręcznej.

   Utwórz plik o nazwie sw.js w folderze głównym i wprowadź zawartość poniższego skryptu. Powodem, dla którego jest zapisywany w katalogu głównym aplikacji, jest umożliwienie mu dostępu do wszystkich plików aplikacji. Dzieje się tak, ponieważ Service Workes ma uprawnienia dostępu tylko do plików w tym samym katalogu i podkatalogach. Jeśli byśmy posiadali jakieś obrazki w filesToCache musielibyśmy je dopisać np. '/images/cat.jpg',

```
const cacheName = 'piac-pwa-v1';

const filesToCache = [
  '/',
  '/index.html',
  '/style.css',
```

```
  '/js/main.js'
];

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

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

self.addEventListener('activate', (event) => {
  const cacheWhitelist = [cacheName];

  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cache) => {
          if (!cacheWhitelist.includes(cache)) {
            return caches.delete(cache);
          }
        })
      );
    })
```

```
  );
});
```

6. Po utworzeniu skryptu sw.js musimy Service Worker zarejestrować w naszej aplikacji. Utwórzmy plik o nazwie main.js w folderze js i wpiszmy następujący kod:

```
window.onload = () => {
 'use strict';

 if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('./sw.js')
   .then(() => console.log('Service Worker registered successfully.'))
   .catch((error) => console.error('Service Worker registration failed:', error));
 }
};
```

7. Możemy także dodać buforowanie dynamiczne w pliku sw.js

```
self.addEventListener('fetch', (event) => {
 event.respondWith(
  caches.match(event.request).then((response) => {
   return response || fetch(event.request).then((fetchResponse) => {
    if (event.request.method === 'GET') {
     return caches.open(cacheName).then((cache) => {
      cache.put(event.request, fetchResponse.clone());
      return fetchResponse;
     });
    }
    return fetchResponse;
   });
  }).catch(() => {
   if (event.request.mode === 'navigate') {
    return caches.match('/index.html');
   }
  })
 );
});
```

8. Ostatnim wymaganiem dla PWA jest posiadanie pliku manifestu. Manifest to plik json , który służy do określania, jak aplikacja będzie wyglądać i zachowywać się na urządzeniach. Na przykład możemy w nim ustawić orientację aplikacji i kolor motywu podobnie jak ma to miejsce w przypadku aplikacji Android. Utwórz plik o nazwie manifest.json w folderze głównym i dodaj następującą zawartość: Opcjonalnie można dodać ikonę favicon, utworzoną w kilku rozmiarach i umieszczoną w folderze image. Ikonę oraz późniejszy splash screen można wygenerować za

pomocą narzędzia: https://www.npmjs.com/package/pwa-asset-generator Ewentualnie do samej ikony można wykorzystać: https://realfavicongenerator.net/

```
{
 "name": "PIAC TEST",
 "short_name": "PIAC",
 "lang": "en-US",
 "start_url": "/index.html",
 "display": "standalone",
 "background_color": "#ffffff",
 "theme_color": "#db4938",
 "icons": [
  {
   "src": "images/pwa-icon-128.png",
   "sizes": "128x128",
   "type": "image/png"
  },
  {
   "src": "images/pwa-icon-144.png",
   "sizes": "144x144",
   "type": "image/png"
  },
  {
   "src": "images/pwa-icon-192.png",
   "sizes": "192x192",
   "type": "image/png"
  },
  {
   "src": "images/pwa-icon-256.png",
   "sizes": "256x256",
   "type": "image/png"
  },
  {
   "src": "images/pwa-icon-512.png",
   "sizes": "512x512",
   "type": "image/png"
  }
 ]
}
```

9. Aby aplikacja działała lepiej w trybie offline, możemy dodać obsługę dynamicznego ładowania treści z pamięci podręcznej. Zaktualizuj sw.js, aby obsługiwać dynamiczne buforowanie:

```
self.addEventListener('fetch', (event) => {
 event.respondWith(
  caches.match(event.request).then((response) => {
   return response || fetch(event.request).then((fetchResponse) => {
    return caches.open(cacheName).then((cache) => {
```

```
        cache.put(event.request, fetchResponse.clone());
        return fetchResponse;
      });
    });
  }).catch(() => {
    // Fallback dla braku połączenia
    if (event.request.mode === 'navigate') {
      return caches.match('/index.html');
    }
  })
 );
});
```

10. Dodaj animacje CSS, aby aplikacja była bardziej atrakcyjna wizualnie. Na przykład:

```
.title {
 font-size: 3rem;
 animation: fadeIn 2s ease-in-out;
}

@keyframes fadeIn {
 from {
  opacity: 0;
  transform: translateY(-20px);
 }
 to {
  opacity: 1;
  transform: translateY(0);
 }
}
```

11. Aby dodać Splash Screen do aplikacji PWA, musimy skonfigurować plik manifestu oraz odpowiednio dostosować ikony i metadane w pliku index.html. Splash Screen pojawia się, gdy użytkownik uruchamia aplikację PWA z ekranu głównego urządzenia mobilnego. Dodaj odpowiednie właściwości do pliku manifest.json, aby określić tło (background_color), kolor motywu (theme_color) oraz ikony w różnych rozmiarach:

```
{
 "name": "PIAC TEST",
 "short_name": "PIAC",
 "lang": "en-US",
 "start_url": "/index.html",
 "display": "standalone",
 "background_color": "#ffffff",
 "theme_color": "#db4938",
 "icons": [
  {
   "src": "images/pwa-icon-128.png",
   "sizes": "128x128",
   "type": "image/png"
  },
  {
```

```
    "src": "images/pwa-icon-144.png",
    "sizes": "144x144",
    "type": "image/png"
   },
   {
    "src": "images/pwa-icon-192.png",
    "sizes": "192x192",
    "type": "image/png"
   },
   {
    "src": "images/pwa-icon-256.png",
    "sizes": "256x256",
    "type": "image/png"
   },
   {
    "src": "images/pwa-icon-512.png",
    "sizes": "512x512",
    "type": "image/png"
   }
 ]
}
```

12. W sekcji <head> pliku index.html dodaj następujące metadane:

```
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="default">
<meta name="apple-mobile-web-app-title" content="PWA PIAC">
<link rel="apple-touch-icon" href="images/pwa-icon-152.png">
```

13. Apple wymaga ręcznego tworzenia obrazów Splash Screen dla różnych rozdzielczości ekranów. Obrazy te muszą być dodane w folderze images, a ich rozmiary powinny odpowiadać poniższym wartościom:

Rozdzielczość obrazu	Urządzenie
640x1136	iPhone SE (1. generacji)
750x1334	iPhone 6/7/8
828x1792	iPhone XR
1125x2436	iPhone X/XS/11 Pro
1242x2688	iPhone XS Max/11 Pro Max
1536x2048	iPad Mini/Air
1668x2224	iPad Pro (10.5 cala)
1668x2388	iPad Pro (11 cali)
2048x2732	iPad Pro (12.9 cala)

Dodaj te obrazy do projektu, a następnie zadeklaruj je w <head>:

<link rel="apple-touch-startup-image" href="images/splash-640x1136.png" media="(device-width: 320px) and (device-height: 568px)">
<link rel="apple-touch-startup-image" href="images/splash-750x1334.png" media="(device-width: 375px) and (device-height: 667px)">
<link rel="apple-touch-startup-image" href="images/splash-828x1792.png" media="(device-width: 414px) and (device-height: 896px)">
<link rel="apple-touch-startup-image" href="images/splash-1125x2436.png" media="(device-width: 375px) and (device-height: 812px)">
<link rel="apple-touch-startup-image" href="images/splash-1242x2688.png" media="(device-width: 414px) and (device-height: 896px)">
<link rel="apple-touch-startup-image" href="images/splash-1536x2048.png" media="(min-device-width: 768px) and (max-device-width: 1024px)">
<link rel="apple-touch-startup-image" href="images/splash-1668x2224.png" media="(min-device-width: 834px) and (max-device-width: 1112px)">
<link rel="apple-touch-startup-image" href="images/splash-1668x2388.png" media="(min-device-width: 834px) and (max-device-width: 1194px)">
<link rel="apple-touch-startup-image" href="images/splash-2048x2732.png" media="(min-device-width: 1024px) and (max-device-width: 1366px)">

14. Rozbuduj aplikację o grafikę, podstrony oraz dodatkowe treści. Tematyka aplikacji jest dowolna np. aplikacja może przedstawiać (nie)śmieszne memy z kotami.

15. Przetestuj czy Splash Screen i aplikacja poprawnie się uruchamia za pomocą narzędzia Lighthouse (wbudowane w Google Chrome DevTools).

16. Wdróż aplikację w dowolnym środowisku chmurowym. Istnieje wiele platform hostingowych, które wspierają HTTPS i są odpowiednie dla aplikacji PWA:

- GitHub Pages
  Darmowa platforma do hostowania statycznych stron internetowych z obsługą HTTPS. Idealna dla projektów open-source lub stron osobistych
- Netlify
  Popularna platforma z darmowym planem, wspierająca automatyczne wdrożenia z repozytoriów GitHub/GitLab oraz rollback do poprzednich wersji.
- Firebase Hosting
  Usługa od Google oferująca bezpieczne hostowanie zarówno statycznych, jak i dynamicznych stron internetowych. Firebase posiada darmowy plan z opcją płatności za zużycie zasobów
- Vercel
  Platforma zoptymalizowana pod kątem nowoczesnych aplikacji webowych (np. Next.js). Oferuje łatwe wdrożenie i obsługę HTTPS
- Lokalny serwer deweloperski
  Do testów można użyć lokalnego serwera HTTP/HTTPS (np. http-server w Node.js lub Apache). Jest to opcja tylko dla środowiska deweloperskiego.

17. Rozważ i spróbuj w grupie dodać funkcje powiadomień (notyfikacji) w aplikacji, które pozwolą użytkownikowi otrzymywać komunikaty nawet wtedy, gdy aplikacja nie jest aktywnie używana. Powiadomienia mogą być przydatne do informowania o nowych treściach,

aktualizacjach lub innych ważnych wydarzeniach. Powiadomienia push wymagają backendu do wysyłania wiadomości do subskrybentów. Możesz użyć narzędzi takich jak Firebase Cloud Messaging (FCM) lub własnego serwera z biblioteką np. web-push dla Node.js.