

Sprawozdanie z tematu IV – Hostowanie aplikacji

Przedmiot: Systemy urządzeń mobilnych

Autor: Jakub Opiełka

Kierunek: Informatyka dla inżynierów

Semestr: I

1. Wstęp

Progressive Web App (PWA) to nowoczesne podejście do tworzenia aplikacji webowych, które łączy cechy tradycyjnych stron internetowych z funkcjonalnościami natywnych aplikacji mobilnych. Dzięki takim technologiom jak Service Worker czy IndexedDB, możliwe jest m.in. działanie offline, szybkie ładowanie czy instalacja na urządzeniu użytkownika.

Celem niniejszego projektu było stworzenie pełnoprawnej aplikacji PWA w czystym JavaScript (bez użycia frameworków), spełniającej określone wymagania funkcjonalne i techniczne. Projekt zakładał implementację własnego service workera, konfigurację manifestu, obsługę trybu offline oraz wdrożenie aplikacji na darmowym serwerze z obsługą HTTPS.

Efektem pracy jest aplikacja WeatherNotes, która umożliwia użytkownikowi sprawdzenie aktualnej pogody w wybranym mieście oraz tworzenie i przeglądanie notatek zapisywanych lokalnie — także w trybie offline.

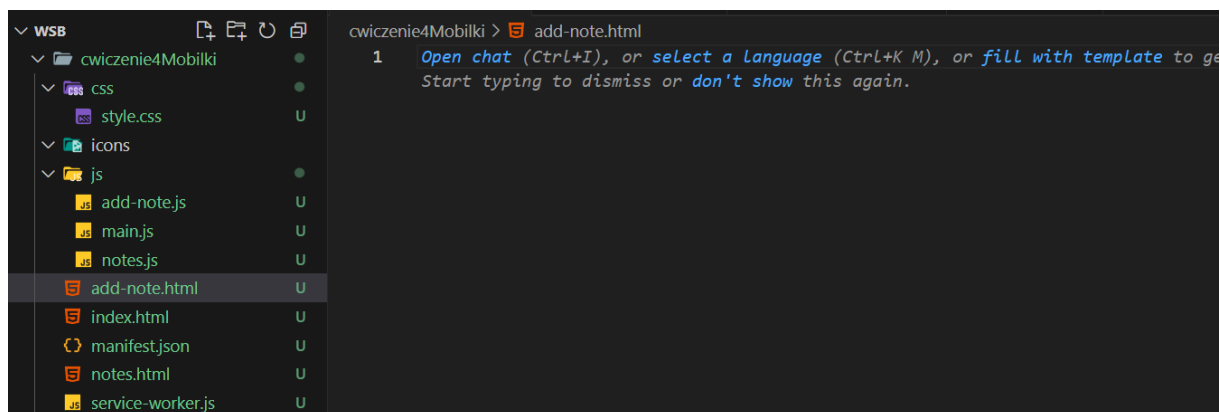
2. Opis funkcjonalności aplikacji.

Aplikacja WeatherNotes łączy dwie praktyczne funkcje: sprawdzanie aktualnej pogody w wybranym mieście oraz zapisywanie własnych notatek, np. dotyczących planów na dany dzień. Użytkownik może:

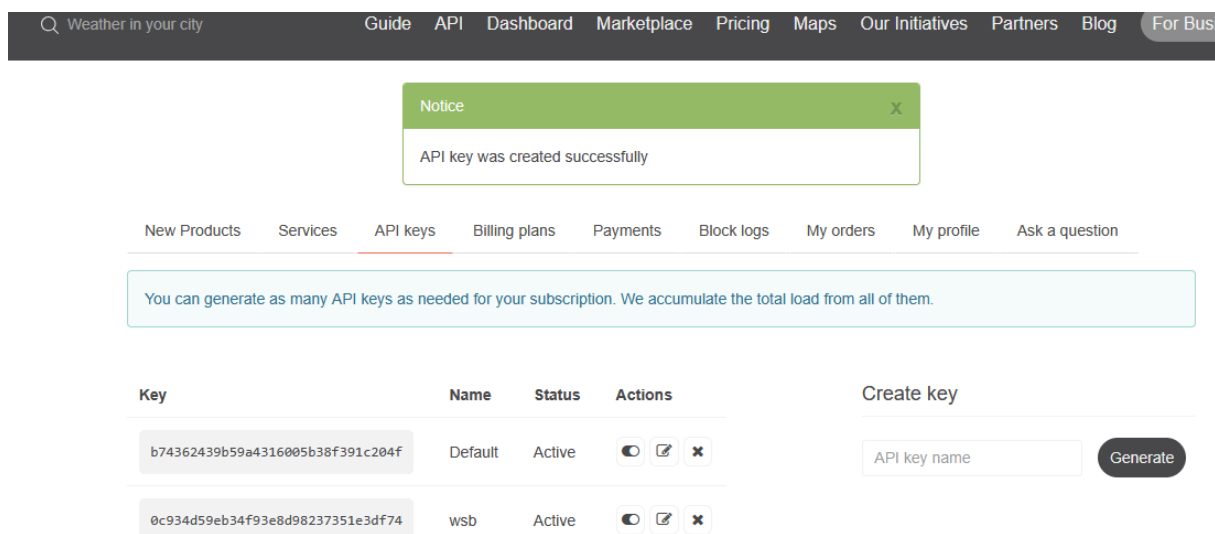
- wyszukać aktualną pogodę na podstawie wpisanego miasta (integracja z API OpenWeatherMap),
- dodać notatkę w prostym formularzu (zapis lokalny w IndexedDB),
- przeglądać i usuwać zapisane notatki,
- korzystać z aplikacji offline – ostatnio pobrana pogoda i zapisane notatki są dostępne bez internetu.

3. Opis techniczny krok po kroku.

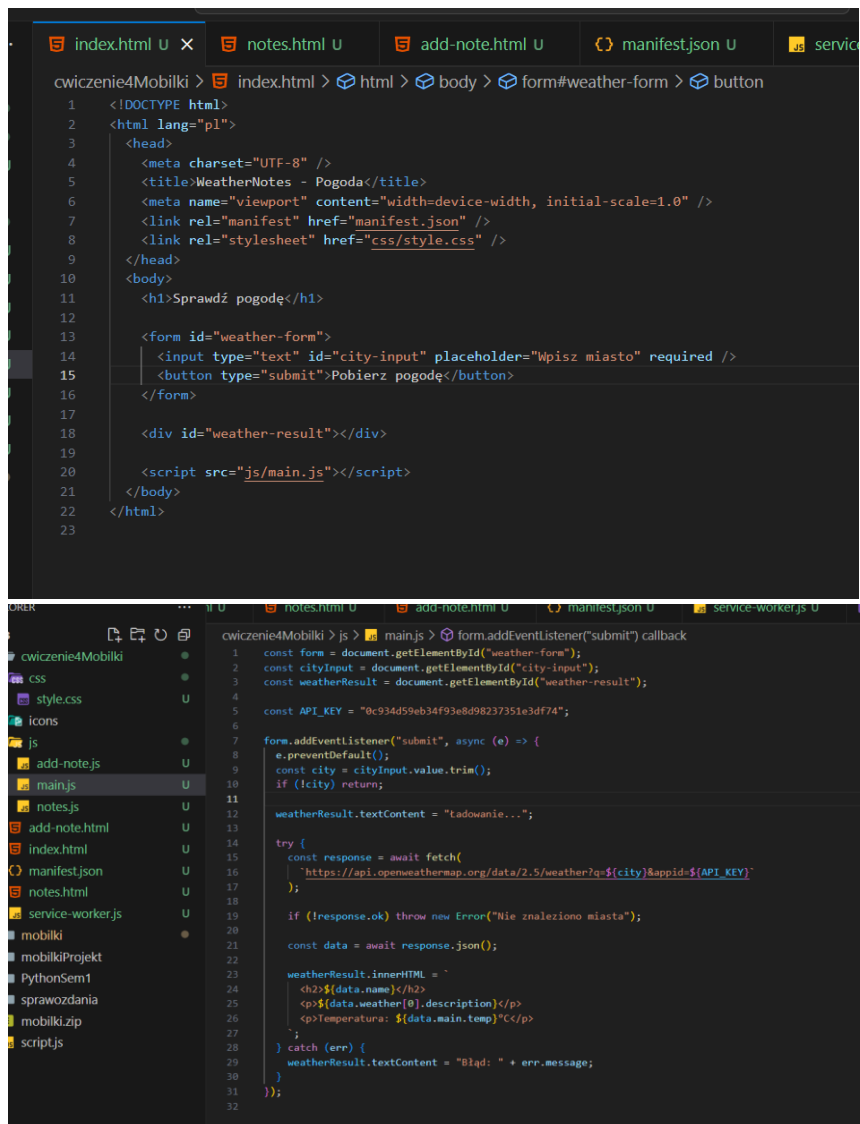
3.1 Tworzymy strukturę projektu w edytorze.



3.2 Logujemy się do serwisu z darmowym API pogodowym – OpenWeather i generujemy klucz do API



3.2 Tworzymy pierwszy panel w pliku index.html oraz łączymy go z logiką w pliku main.js, która uderza od razu do tego API.



The screenshot shows a VS Code editor with two files open: `index.html` and `main.js`. The `index.html` file contains the following code:

```
1 <!DOCTYPE html>
2 <html lang="pl">
3   <head>
4     <meta charset="UTF-8" />
5     <title>WeatherNotes - Pogoda</title>
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <link rel="manifest" href="manifest.json" />
8     <link rel="stylesheet" href="css/style.css" />
9   </head>
10  <body>
11    <h1>Sprawdź pogodę</h1>
12
13    <form id="weather-form">
14      <input type="text" id="city-input" placeholder="Wpisz miasto" required />
15      <button type="submit">Pobierz pogodę</button>
16    </form>
17
18    <div id="weather-result"></div>
19
20    <script src="js/main.js"></script>
21  </body>
22 </html>
23
```

The `main.js` file contains the following code:

```
1 const form = document.getElementById("weather-form");
2 const cityInput = document.getElementById("city-input");
3 const weatherResult = document.getElementById("weather-result");
4
5 const API_KEY = "0c934d59eb34f93e8d98237351e3df74";
6
7 form.addEventListener("submit", async (e) => {
8   e.preventDefault();
9   const city = cityInput.value.trim();
10  if (!city) return;
11
12  weatherResult.textContent = "Ładowanie...";
13
14  try {
15    const response = await fetch(
16      `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API_KEY}`
17    );
18    if (!response.ok) throw new Error("Nie znaleziono miasta");
19    const data = await response.json();
20
21    weatherResult.innerHTML = `
22      <h2>${data.name}</h2>
23      <p>${data.weather[0].description}</p>
24      <p>Temperatura: ${data.main.temp}°C</p>
25    `;
26  } catch (err) {
27    weatherResult.textContent = "Błąd: " + err.message;
28  }
29 });
```

Mamy pierwszy sukces, dane zostały zwrócone:



3.3 Implementujemy formularz a więc pliki add-note.html oraz add-note.js, warto zauważyć, że wprowadzamy tutaj indexedDB

The screenshot displays two side-by-side code editors. The left editor shows a JavaScript file named `add-note.js` with the following code:

```
1 let db;
2 
3 const request = indexedDB.open("WeatherNotesDB", 1);
4 
5 request.onerror = () => {
6   console.error("Błąd podczas otwierania IndexedDB");
7 };
8 
9 request.onsuccess = () => {
10   db = request.result;
11 };
12 
13 request.onupgradeneeded = (event) => {
14   db = event.target.result;
15   const store = db.createObjectStore("notes", { keyPath: "id", autoIncrement: true });
16   store.createIndex("title", "title", { unique: false });
17 };
18 
19 document.getElementById("note-form").addEventListener("submit", e => {
20   e.preventDefault();
21 
22   const title = document.getElementById("note-title").value;
23   const content = document.getElementById("note-content").value;
24 
25   const tx = db.transaction("notes", "readwrite");
26   const store = tx.objectStore("notes");
27 
28   const note = {
29     title,
30     content,
31     timestamp: new Date(),
32   };
33 
```

The right editor shows an HTML file named `add-note.html` with the following code:

```
1 <!DOCTYPE html>
2 <html lang="pl">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Dodaj notatkę</title>
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <link rel="stylesheet" href="css/style.css" />
8   </head>
9   <body>
10    <h1>Dodaj notatkę</h1>
11 
12    <form id="note-form">
13      <label for="note-title">Tytuł:</label><br />
14      <input type="text" id="note-title" required /><br />
15 
16      <label for="note-content">Treść:</label><br />
17      <textarea id="note-content" required/></textarea><br />
18 
19      <button type="submit">Zapisz</button>
20    </form>
21 
22    <p id="status"></p>
23 
24    <a href="notes.html">Zobacz zapisane notatki</a>
25 
26    <script src="js/add-note.js"></script>
27  </body>
28 </html>
```

Mamy kolejny sukces, po wprowadzeniu notatki dostajemy komunikat o zapisie:

Dodaj notatkę

Tytuł:

Treść:

Notatka zapisana!

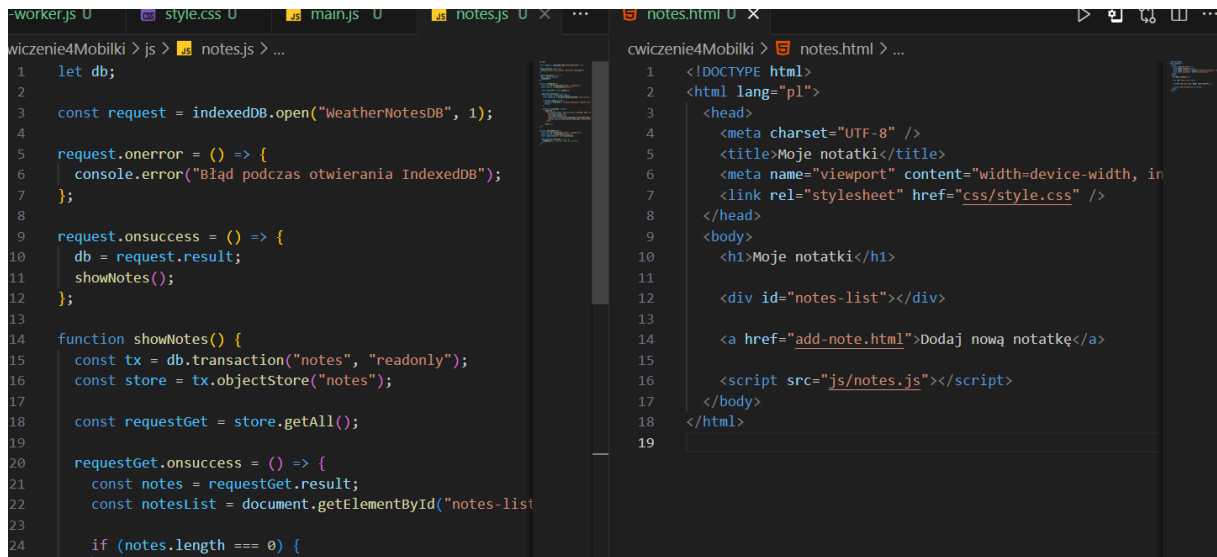
[Zobacz zapisane notatki](#)

Po przejściu do Application – IndexedDB, widzimy zapisane dane:

The screenshot shows the VS Code interface with the Chrome DevTools console open. The 'Application' tab is selected, displaying a table of data. The table has two columns: 'Key (Key path: "id")' and 'Value'. The first row shows a key of '0' and a value of '0'. The second row shows a key of '1' and a value of a JSON object: `{title: 'Jaki tytul', contents: 'Jakas notatka', timestamp: Fri May 16 2025 22:37:44 GMT+0...}`. The left sidebar shows the file explorer with a project structure including 'Local storage', 'Session storage', 'Extension storage', 'IndexedDB', and 'WeatherNotesDB'.

Key (Key path: "id")	Value
0	0
1	<code>{title: 'Jaki tytul', contents: 'Jakas notatka', timestamp: Fri May 16 2025 22:37:44 GMT+0...}</code>

3.4 Implementujemy panel note, co istotne, dane pobieramy już z naszej indexedDB, te dane które były wcześniej przedstawione na obrazku w narzędziach przeglądarki.

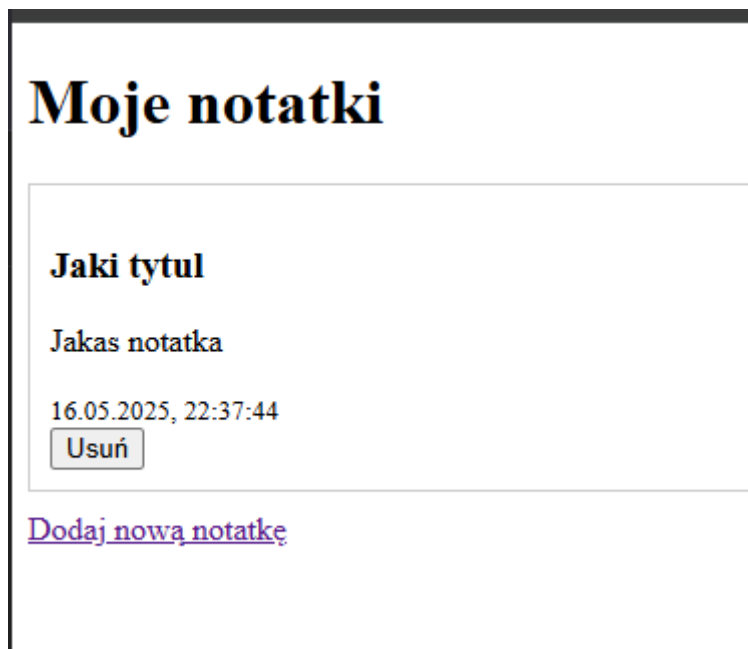


```
worker.js U | style.css U | main.js U | notes.js U | ... | notes.html U X
wyczenie4Mobilki > js > notes.js > ...
1 let db;
2
3 const request = indexedDB.open("WeatherNotesDB", 1);
4
5 request.onerror = () => {
6   console.error("Błąd podczas otwierania IndexedDB");
7 };
8
9 request.onsuccess = () => {
10   db = request.result;
11   showNotes();
12 };
13
14 function showNotes() {
15   const tx = db.transaction("notes", "readonly");
16   const store = tx.objectStore("notes");
17
18   const requestGet = store.getAll();
19
20   requestGet.onsuccess = () => {
21     const notes = requestGet.result;
22     const notesList = document.getElementById("notes-list");
23
24     if (notes.length === 0) {
```

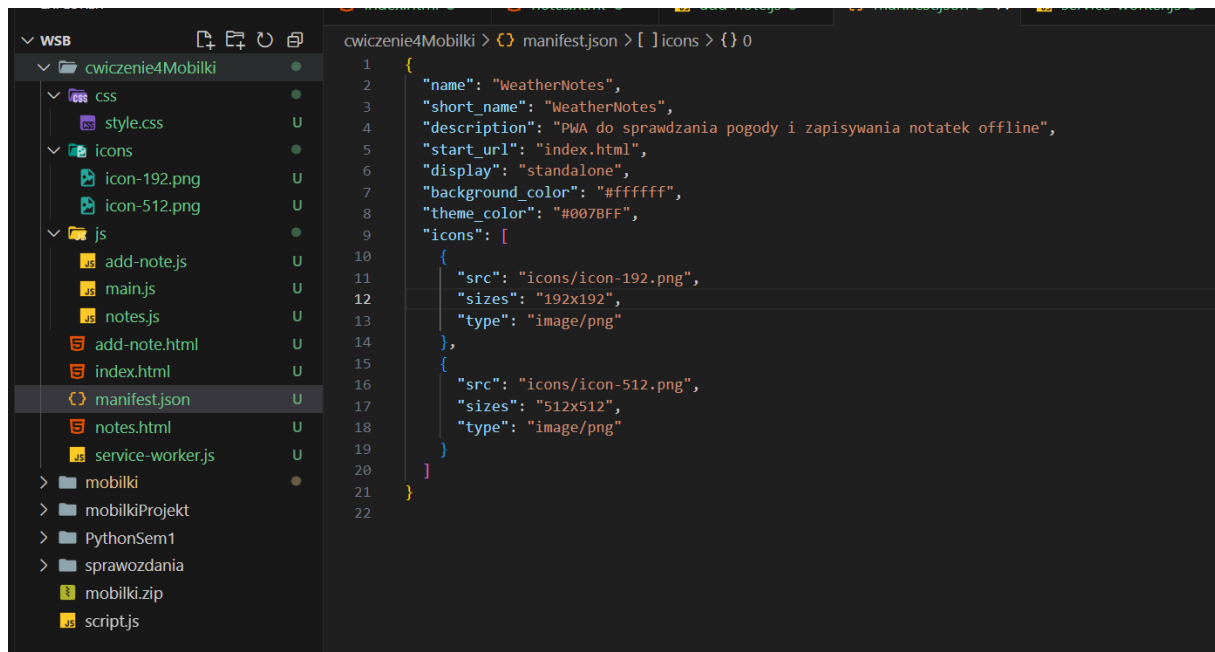
```

wyczenie4Mobilki > notes.html > ...
1 <!DOCTYPE html>
2 <html lang="pl">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Moje notatki</title>
6     <meta name="viewport" content="width=device-width, in
7     <link rel="stylesheet" href="css/style.css" />
8   </head>
9   <body>
10     <h1>Moje notatki</h1>
11
12     <div id="notes-list"></div>
13
14     <a href="add-note.html">Dodaj nową notatkę</a>
15
16     <script src="js/notes.js"></script>
17   </body>
18 </html>
19
```

Panel rzeczywiście wyświetla naszą notatkę, działa również routing do formularza dodawania:



3.5 Implementujemy manifest.json oraz dodajemy ikony. Plik manifest.json zawiera metadane aplikacji PWA, które pozwalają przeglądarce rozpoznać ją jako aplikację „instalowalną”. Dzięki temu użytkownik może dodać aplikację do ekranu głównego tak jak natywną aplikację mobilną. W pliku zdefiniowano m.in. nazwę, kolory, ikonki w wymaganych rozdzielczościach oraz stronę startową. Manifest jest podłączony do każdej strony aplikacji za pomocą znacznika `<link rel="manifest">` w sekcji `<head>`.



3.6 Pora na service worker, a więc mózg naszej aplikacji PWA. W service-worker.js zastosowano strategię cache-first dla plików statycznych (HTML, CSS, JS, ikony), aby aplikacja mogła działać offline. Dla danych z zewnętrznego API (OpenWeather) użyto strategii network-first, by zawsze próbować pobrać najnowsze dane. Pliki są cache'owane podczas instalacji, a stare cache'y usuwane przy aktywacji, co zapobiega błędom przy aktualizacji aplikacji.

Dodatkowo notatki użytkownika są przechowywane lokalnie w przeglądarce za pomocą IndexedDB i nie są przesyłane do żadnego serwera, co zapewnia prywatność oraz dostępność w trybie offline.

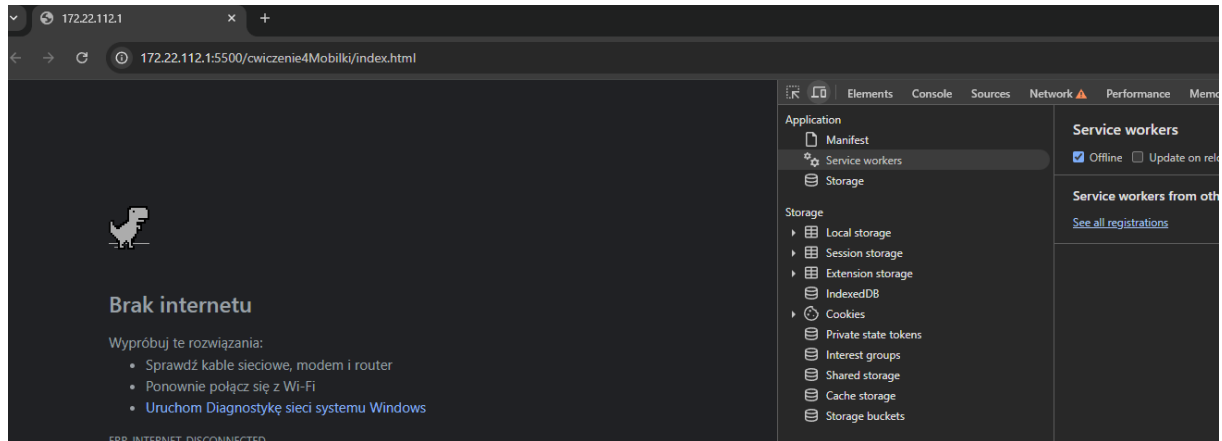
```

1  const CACHE_NAME = 'weathernotes-cache-v1';
2  const STATIC_ASSETS = [
3    '/',
4    '/index.html',
5    '/add-note.html',
6    '/notes.html',
7    '/css/style.css',
8    '/js/main.js',
9    '/js/add-note.js',
10   '/js/notes.js',
11   '/manifest.json',
12   '/icons/icon-192.png',
13   '/icons/icon-512.png'
14 ];
15
16 // Instalacja - cache plików statycznych
17 self.addEventListener('install', event => {
18   event.waitUntil(
19     caches.open(CACHE_NAME)
20       .then(cache => cache.addAll(STATIC_ASSETS))
21   );
22 });
23
24 // Aktywacja - czyszczenie starych cache
25 self.addEventListener('activate', event => {
26   event.waitUntil(
27     caches.keys().then(keys =>
28       Promise.all(keys.filter(k => k !== CACHE_NAME).map(k => caches.delete(k)))
29     )
30   );
31 });
32
33 // Obsługa żądań - cache-first dla statycznych, network-first dla API
34 self.addEventListener('fetch', event => {
35   const url = new URL(event.request.url);
36
37   if (url.origin === location.origin) {
38     // Cache-first dla lokalnych zasobów
39     event.respondWith(
40       caches.match(event.request).then(cached =>
41         cached || fetch(event.request).catch(() => offlineFallback(url.pathname))
42       )
43     );
44   } else {
45     // Network-first dla API
46     event.respondWith(
47       fetch(event.request)
48         .then(response => response)
49         .catch(() => new Response('Brak internetu i brak danych z cache.', { status: 503 }));
50     );
51   }
52 });
53
54 function offlineFallback(pathname) {
55   if (pathname.endsWith('.html')) {
56     return caches.match('/index.html');
57   }
58   return new Response('Brak internetu', { status: 503 });
59 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AZURE

Niestety tym razem nie mamy sukcesu, service worker nie jest wykrywany:



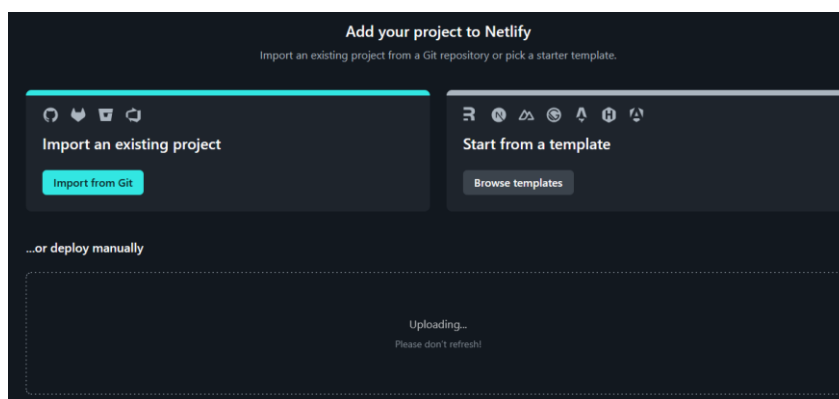
Po dodaniu skryptu rejestrującego dodatkowo manualnie service worker, niestety nadal bez sukcesu:



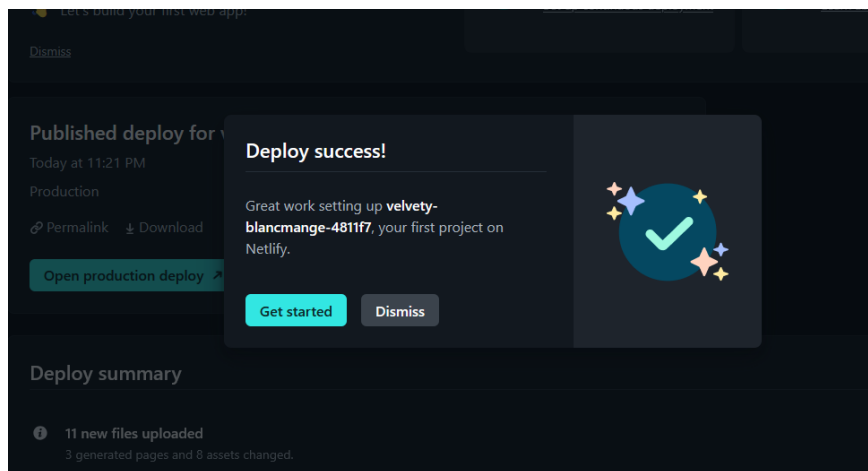
Być może to kwestia uruchamiania przez live server, dlatego przejdziemy do hostowania aplikacji na netlify, być może tam service worker zadziała.

3.7 Deployment na netlify.

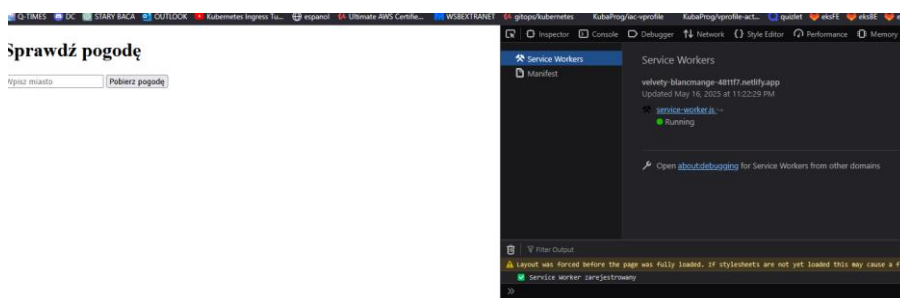
Pakujemy pliki naszego projektu w plik .zip i wrzucamy do Netlify:



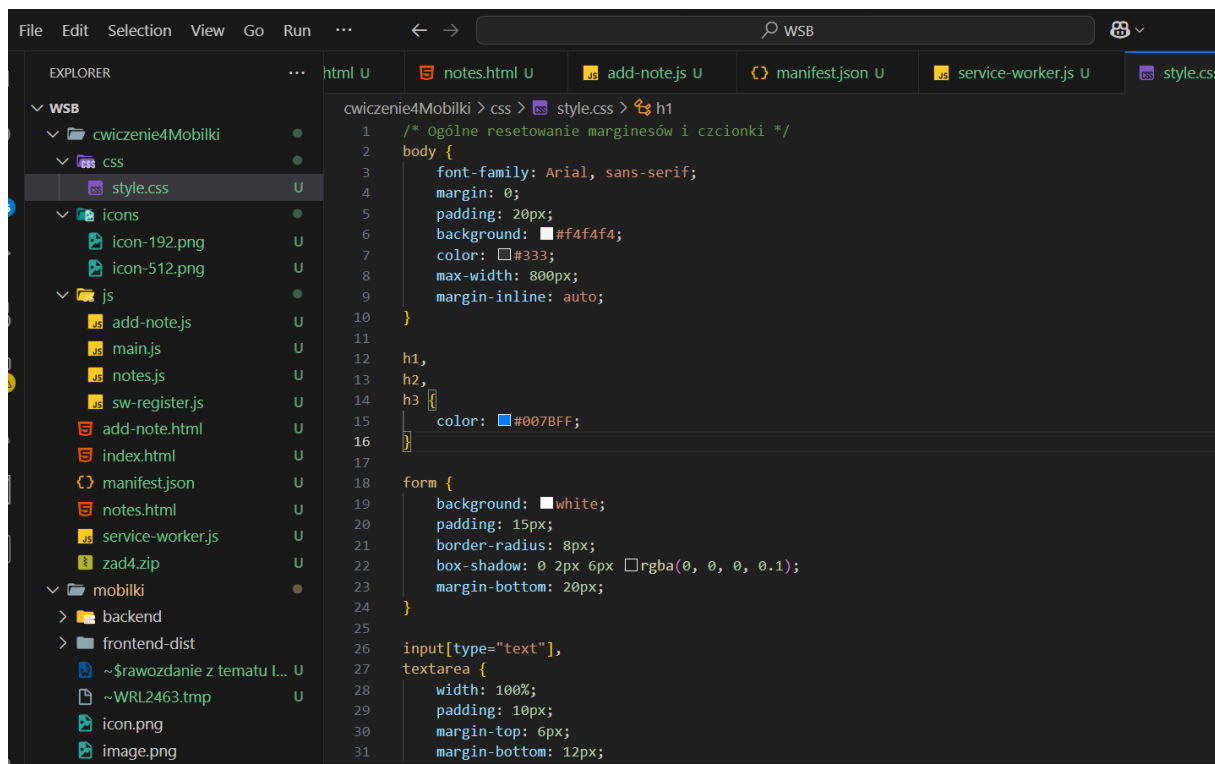
Wygląda na to, że udało się zhostować:



Deployment pomógł, service worker jest teraz widziany:



3.8 Aplikacja ma już zaimplementowane większość rzeczy, ale aby aplikacja spełniała jak najlepiej standardy PWA, dodajemy style, mając na uwadze również responsywność.

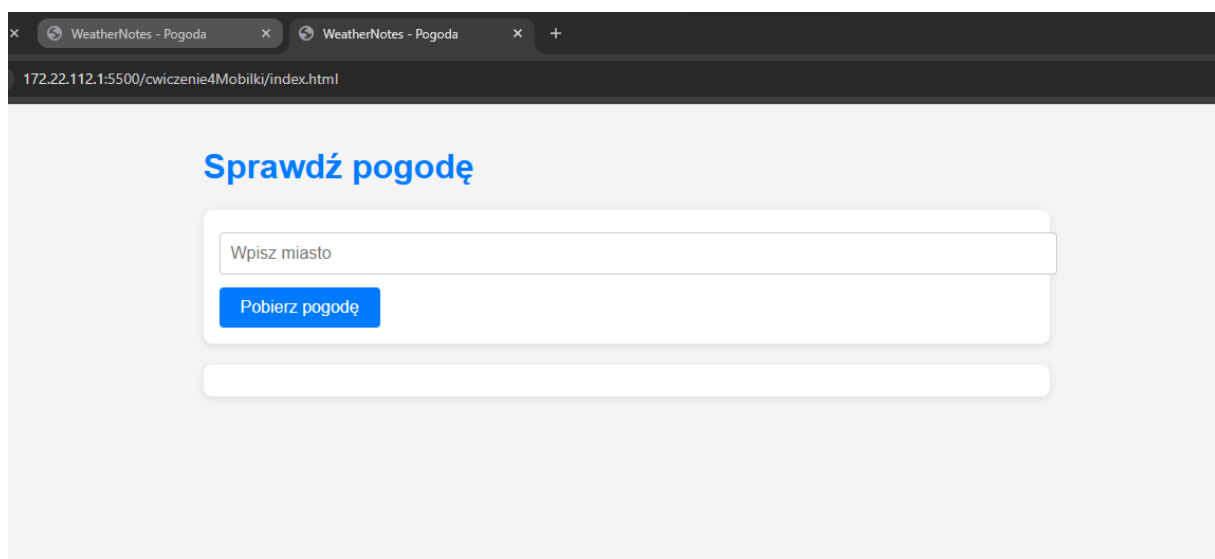


The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left and the Editor window on the right. The Explorer sidebar shows the project structure for 'WSB' and 'cwiczenie4Mobilki'. The Editor window displays the 'style.css' file with the following content:

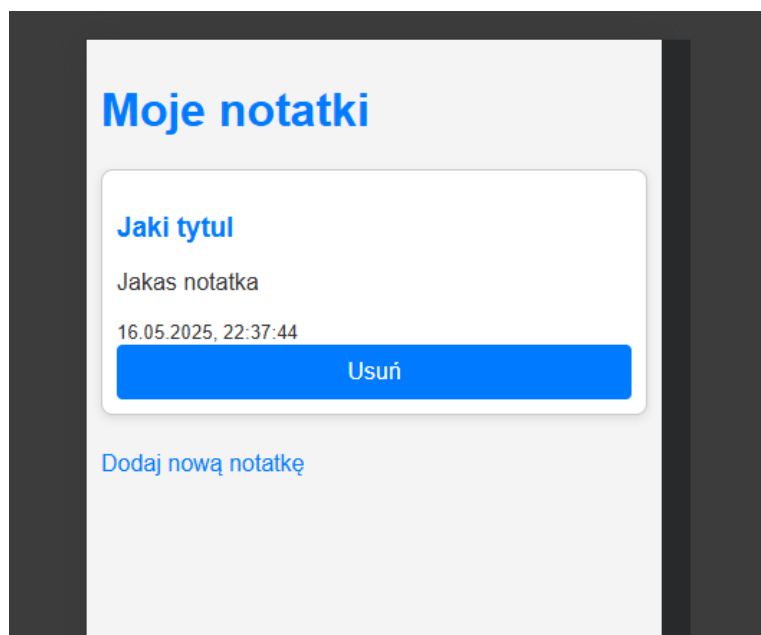
```
1  /* Ogólne resetowanie marginesów i czcionki */
2  body {
3      font-family: Arial, sans-serif;
4      margin: 0;
5      padding: 20px;
6      background-color: #f4f4f4;
7      color: #333;
8      max-width: 800px;
9      margin-left: auto;
10 }
11
12 h1,
13 h2,
14 h3 {
15     color: #007bff;
16 }
17
18 form {
19     background: white;
20     padding: 15px;
21     border-radius: 8px;
22     box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
23     margin-bottom: 20px;
24 }
25
26 input[type="text"],
27 textarea {
28     width: 100%;
29     padding: 10px;
30     margin-top: 6px;
31     margin-bottom: 12px;
```

W rezultacie, aplikacja zaczęła wyglądać o wiele lepiej:

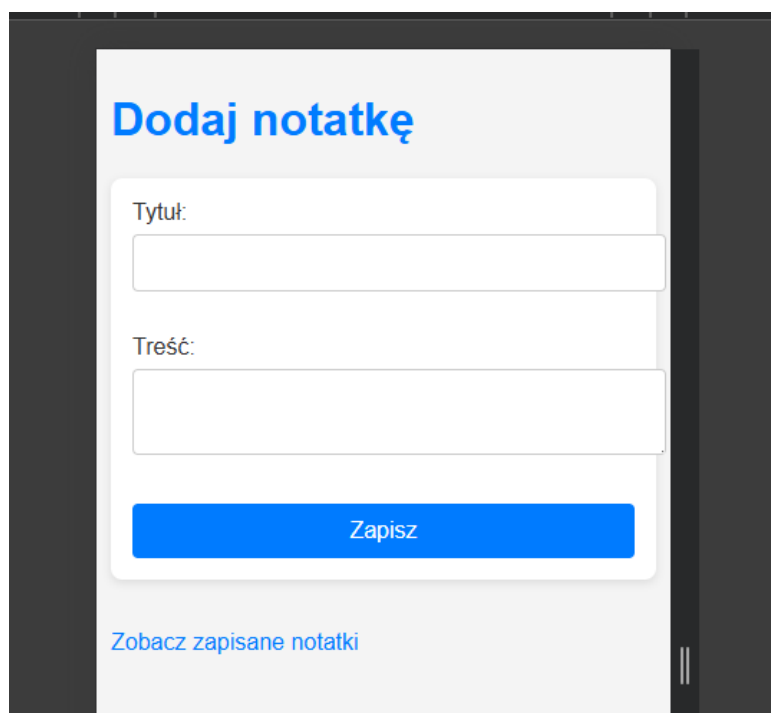
Sprawdzanie pogody:



Lista notatek z ekranu telefonu:



Widok formularza nowej notatki:



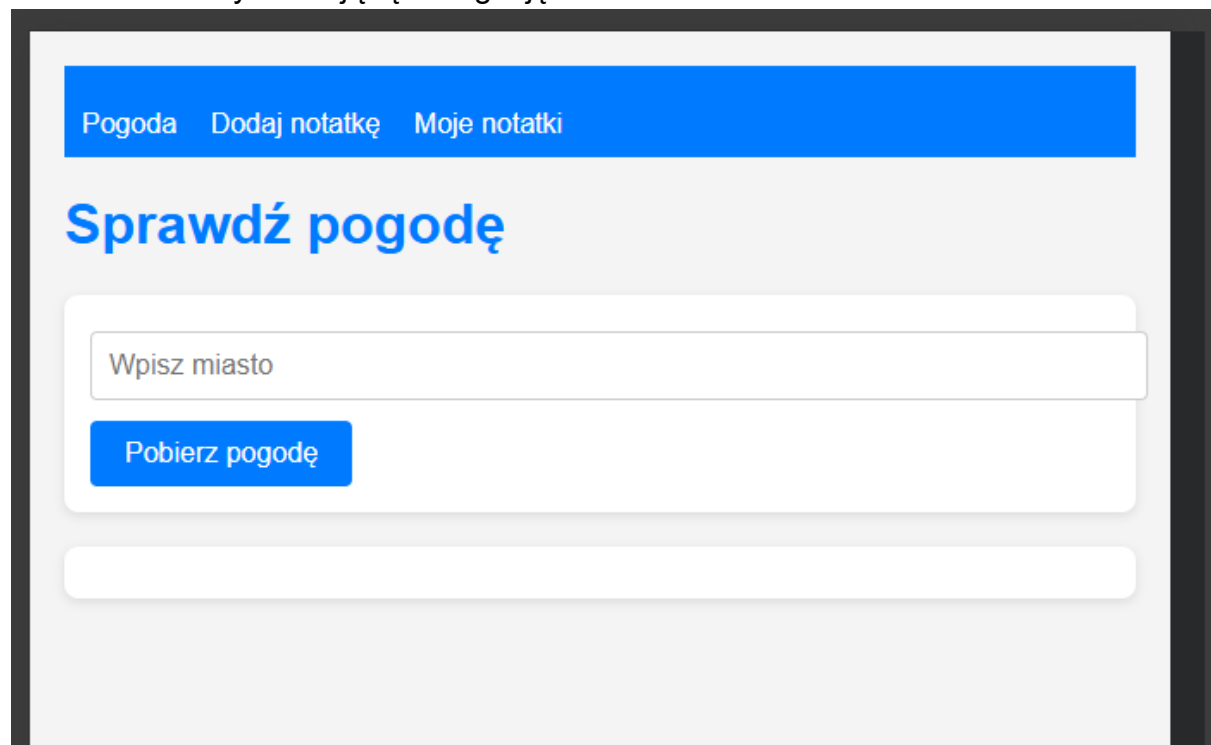
Aby aplikacja miała możliwość nawigacji między panelem pogodowym i notatkowym, dodajemy dodatkową nawigację:

Dodajemy sekcję nawigacji na górze każdego widoku, czyli na początku sekcji body:

```
</head>
<body>
  <nav style="background: #007bff; padding: 10px">
    <a href="index.html" style="color: white; margin-right: 15px">Pogoda</a>
    <a href="add-note.html" style="color: white; margin-right: 15px"
      >Dodaj notatkę</a>
    <a href="notes.html" style="color: white">Moje notatki</a>
  </nav>

  <h1>Moje notatki</h1>
```

W efekcie mamy działającą nawigację:



4. Trudności i ich rozwiązania

4.1. Service Worker nie rejestrował się lokalnie

Początkowo aplikacja nie rejestrowała poprawnie pliku service-worker.js podczas pracy z lokalnym serwerem Live Server. W konsoli przeglądarki nie pojawiały się żadne logi rejestracji, a testowa wizyta pod adresem /service-worker.js zwracała komunikat Cannot GET. Problem wynikał z faktu, że Live Server uruchamiał aplikację z podkatalogu (/weather-notes/), a ścieżki w Service Workerze były zdefiniowane względem root (/). Dodatkowo, Live Server nie obsługuje Service Workerów w sposób w pełni zgodny z produkcyjnym środowiskiem. Problem został rozwiązany poprzez wdrożenie aplikacji na Netlify, które oferuje HTTPS i pełne wsparcie dla PWA. Nauczyło mnie to, że testowanie PWA wymaga odpowiedniego środowiska, najlepiej zbliżonego do rzeczywistego hostingu.

4.2. Netlify przy routingu kierował na /add-note zamiast na /add-note.html, stąd strona offline nie wczytywała się poprawnie, aby rozwiązać ten problem należało zupełnie zmienić podejście do nawigacji, zamiast używać klasycznego href w znacznikach a, należało użyć następującej funkcji:

```
function navigateTo(page) window.location.href = page }
```

4.3. IndexedDB nie inicjalizował się poprawnie

Podczas tworzenia formularza do zapisu notatek pojawiał się problem braku połączenia z bazą danych IndexedDB. Po analizie okazało się, że kod uruchamiający transakcję do zapisu był wykonywany zanim baza danych została zainicjalizowana przez indexedDB.open(...). Problem został rozwiązany poprzez przeniesienie logiki zapisu wewnątrz request.onsuccess, co zagwarantowało, że baza danych była w pełni gotowa przed wykonaniem dalszych operacji. Nauczyło mnie to, że praca z IndexedDB wymaga asynchronicznego podejścia i cierpliwego czekania na inicjalizację.

5. Podsumowanie:

Aplikacja działa zgodnie z założeniami — poprawnie wyświetla dane pogodowe z zewnętrznego API, umożliwia dodawanie i przeglądanie notatek offline dzięki IndexedDB oraz spełnia wymagania PWA: posiada manifest, ikony, service workera i działa po zainstalowaniu jak natywna aplikacja.

Wymagałaby dopracowania pod kątem UX, np. lepszych komunikatów dla użytkownika czy informacji o stanie offline/online. Dodatkowo można byłoby rozważyć integrację z backendem lub opcję eksportu notatek.

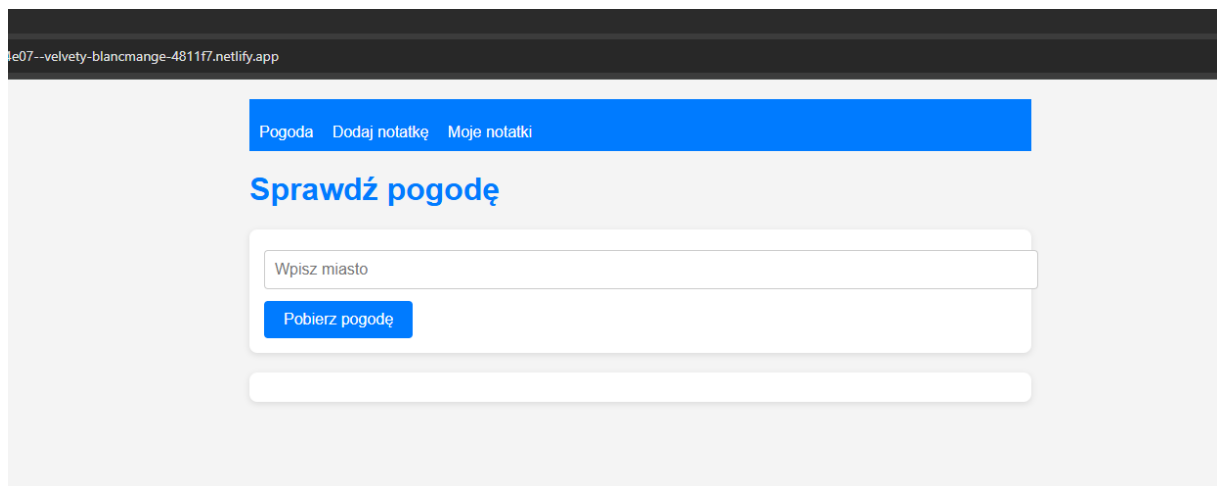
W trakcie pracy nauczyłem się praktycznego wykorzystania Service Workera i IndexedDB, zrozumiałem ograniczenia środowisk developerskich takich jak Live Server oraz dowiedziałem się, jak prawidłowo przygotować i wdrożyć aplikację PWA na produkcyjny hosting.

6. Załączniki

LINK do netlify: <https://monumental-salmiakki-ba3743.netlify.app/>

LINK do github: <https://github.com/KubaProg/wsbCwiczenie4Mobilki>

Zrzuty online:



177.netlify.app/add-note

Pogoda Dodaj notatkę Moje notatki

Dodaj notatkę

Tytuł:

WSB cwiczenie 4

Treść:

opis notatki cwiczenie 4

Zapisz

[Zobacz zapisane notatki](#)

Pogoda Dodaj notatkę Moje notatki

Sprawdź pogodę

warszawa

Pobierz pogodę

Warsaw

overcast clouds

Temperatura: 278.97°C

Zrzuty offline:

WeatherNotes - Moje notatki

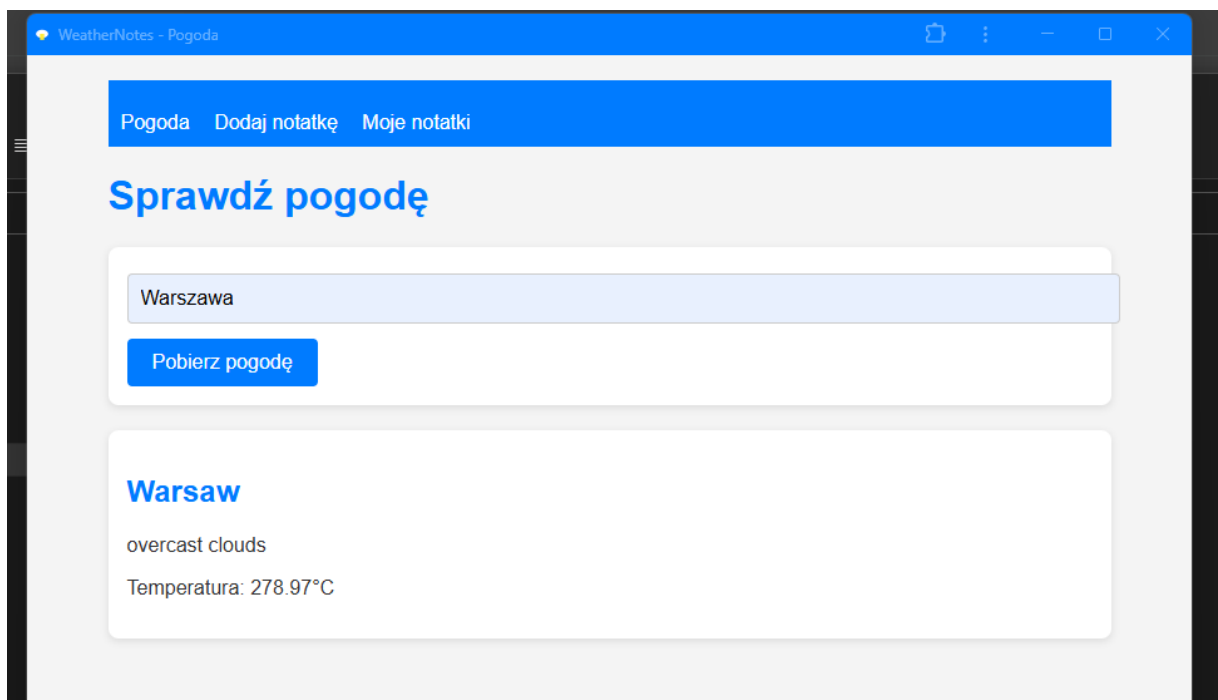
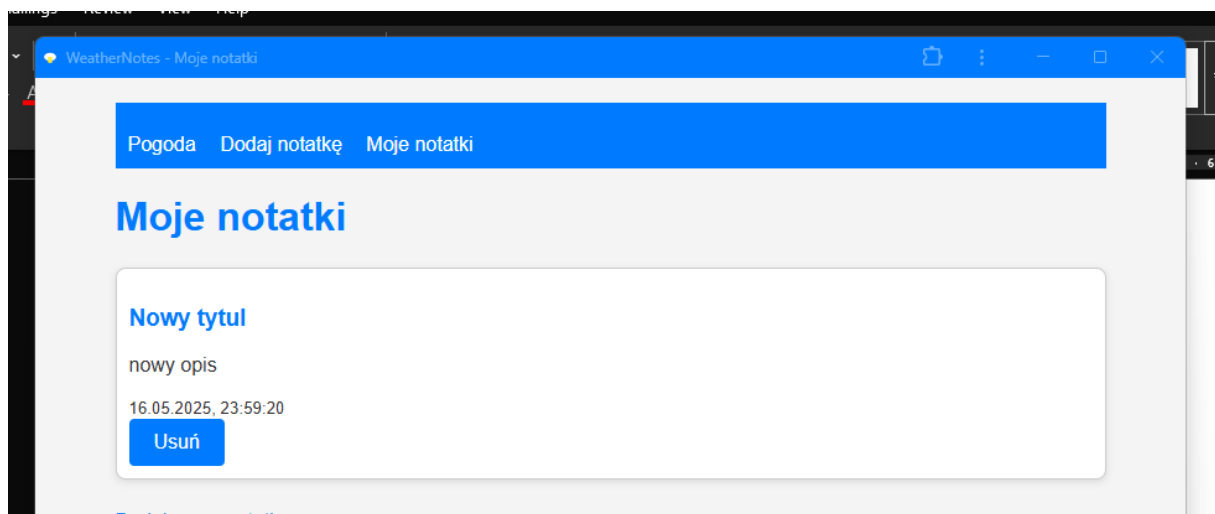
Pogoda Dodaj notatkę Moje notatki

Moje notatki

Brak zapisanych notatek.

[Dodaj nową notatkę](#)

W



Wynik z Lighthouse (SEO nisko, ale zakładam iż nie jest tu istotne)

