SPRAWOZDANIE

Zadanie 4 – Stwórz i opublikuj własną aplikację PWA

Adam Zajler, 131892

chx94178@student.chorzow.merito.pl

Spis treści

1. Wstęp

1.1. Wprowadzenie do Progressive Web Apps (PWA)

Progresywne Aplikacje Internetowe (PWA) to nowoczesne podejście do tworzenia aplikacji webowych, które łączą najlepsze cechy stron internetowych i aplikacji natywnych. Dzięki wykorzystaniu nowoczesnych technologii webowych, takich jak Service Workery, Local Storage, IndexedDB, PWA oferują użytkownikom doświadczenia zbliżone do aplikacji instalowanych na urządzeniach mobilnych.
Kluczowe cechy PWA to m.in. responsywność, możliwość pracy w trybie offline, instalowalność na ekranie głównym urządzenia oraz wysoka wydajność.

1.2. Cel projektu

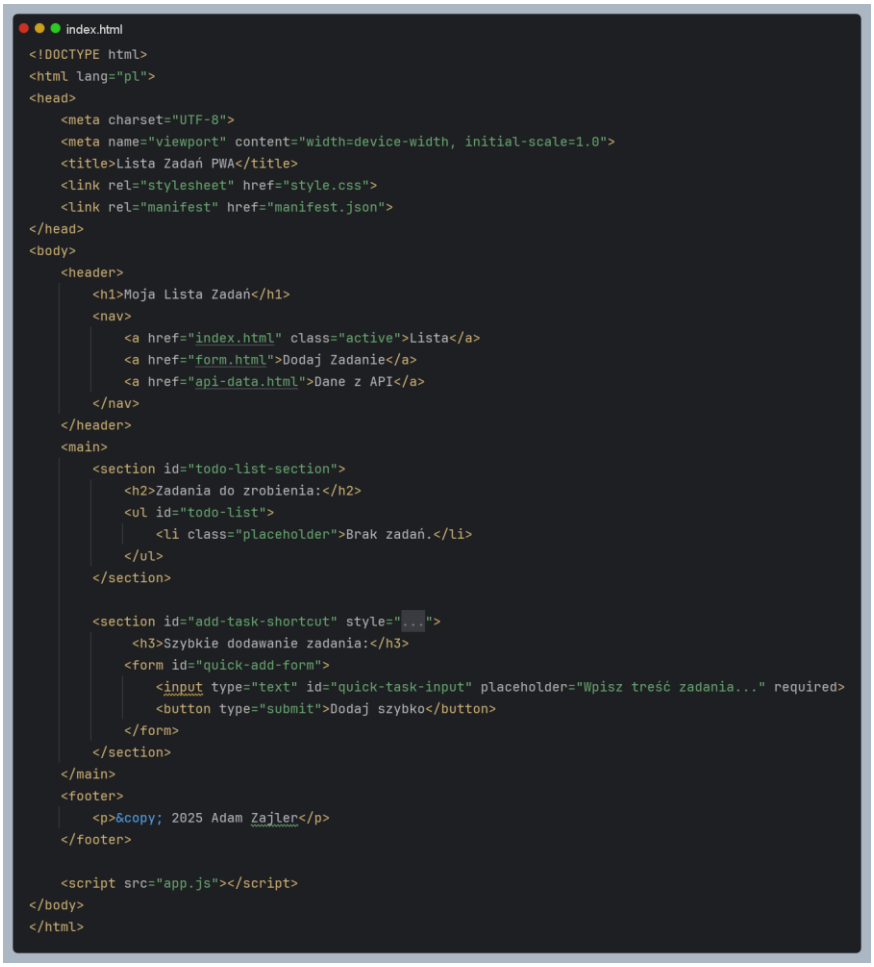
Celem niniejszego projektu było zaprojektowanie, zaimplementowanie oraz wdrożenie pełnoprawnej aplikacji PWA typu "Lista Zadań" (ToDo List). Aplikacja została stworzona w czystym języku JavaScript, bez użycia frameworków jak React, Vue czy Angular.
Projekt zakładał implementację kluczowych mechanizmów PWA, w tym pliku manifestu, własnego Service Workera, wsparcia dla trybu offline poprzez buforowanie zasobów, integrację z IndexedDB do przechowywania danych lokalnie oraz interakcję z zewnętrznym API. Końcowym etapem było hostowanie aplikacji na darmowym serwerze oraz przygotowanie szczegółowego sprawozdania dokumentującego proces realizacji.

2. Opis Funkcjonalności Aplikacji

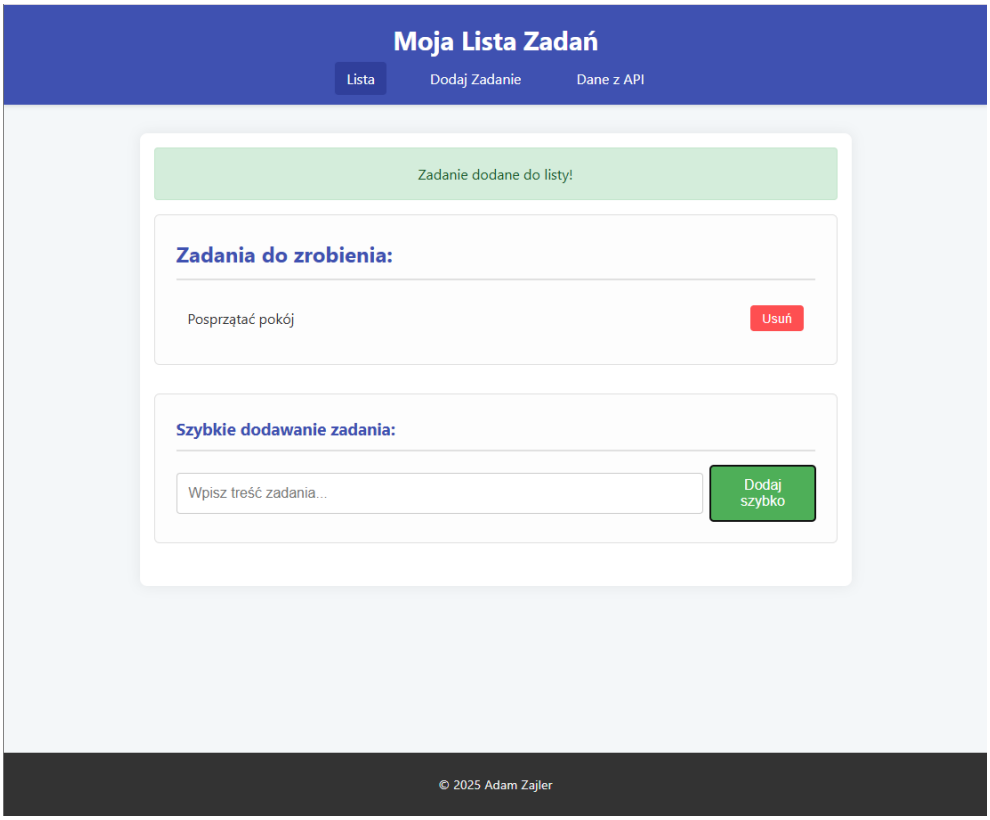
2.1. Główne funkcje

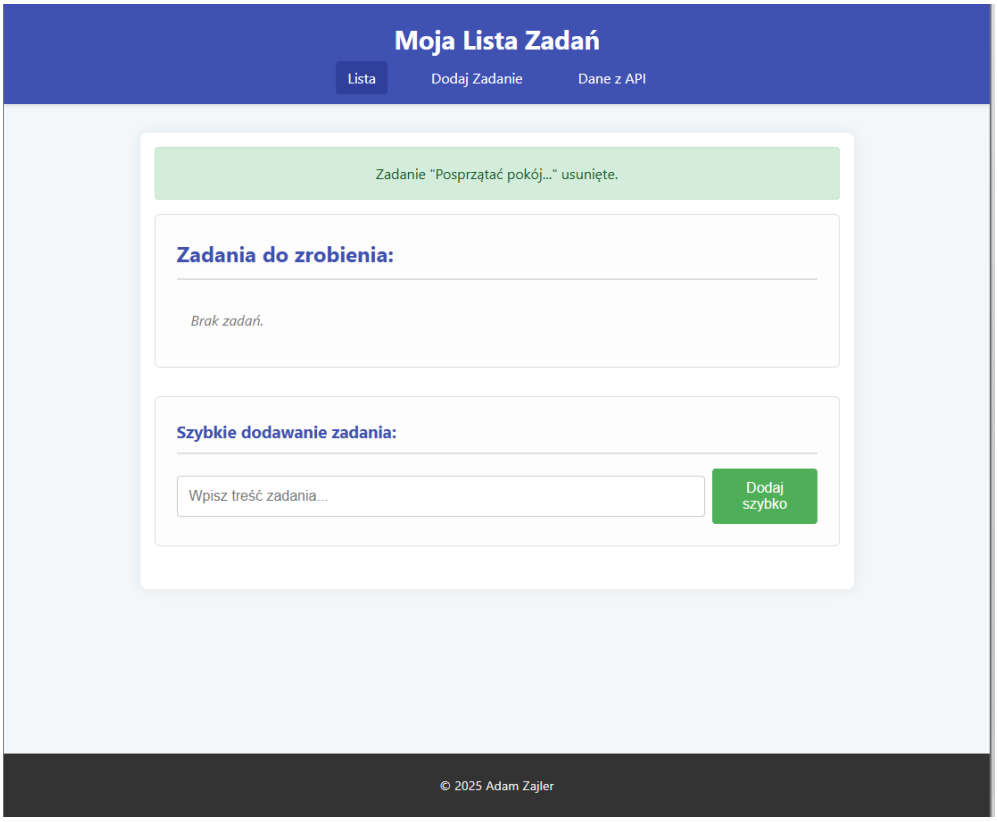
Aplikacja została zaprojektowana jako proste, ale funkcjonalne narzędzie do zarządzania codziennymi zadaniami. Główne funkcje aplikacji obejmują:

1) Zarządzanie zadaniami (Strona główna - index.html):
   a) Wyświetlanie listy zadań pobranych z lokalnej bazy danych IndexedDB.
   b) Możliwość dodawania nowych zadań za pomocą formularza "szybkiego dodawania" bezpośrednio na stronie głównej.
   c) Oznaczanie zadań jako ukończone (wizualne przekreślenie, aktualizacja statusu w IndexedDB).
   d) Usuwanie zadań z listy (oraz z IndexedDB).
   e) Wyświetlanie informacji o terminie i opisie zadania (jeśli zostały dodane).

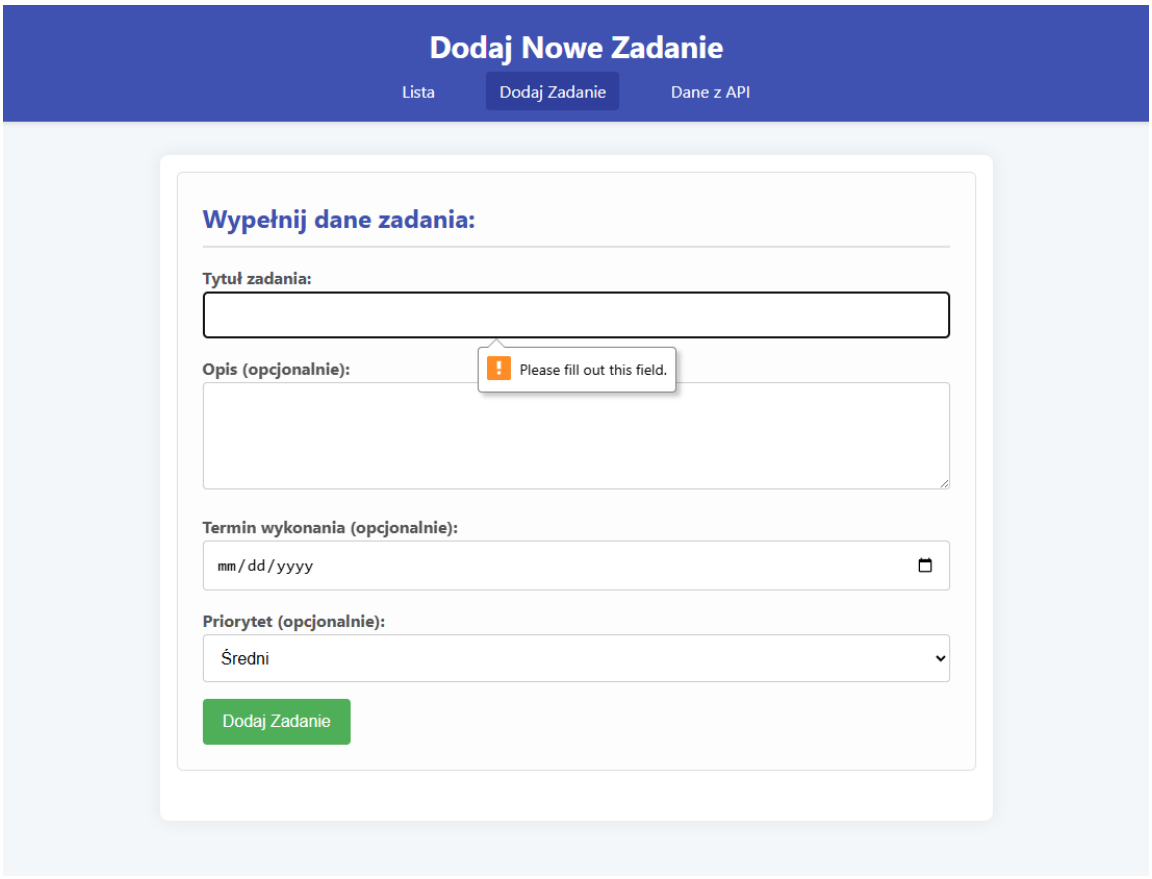

index.html

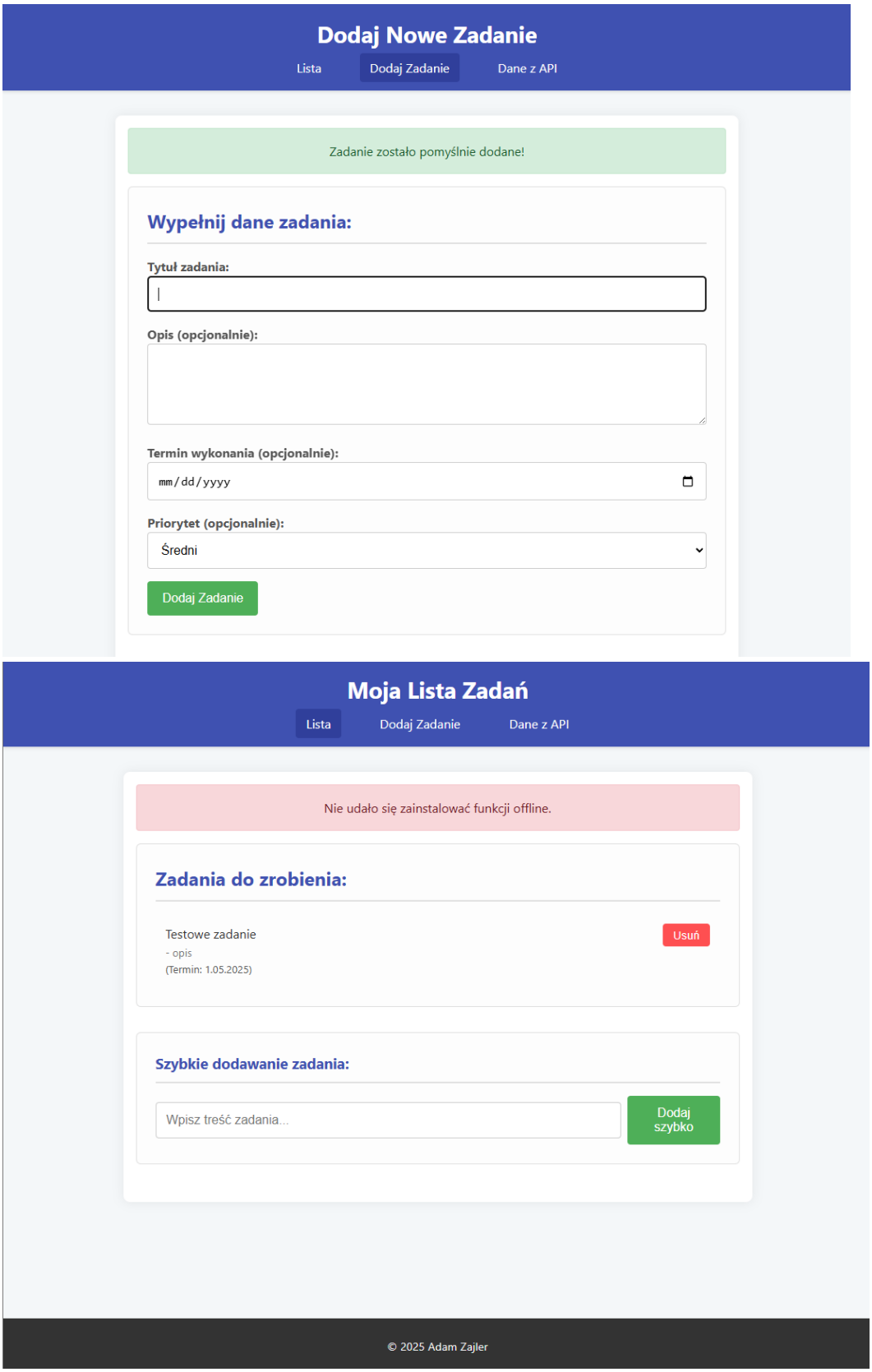
```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista Zadań PWA</title>
    <link rel="stylesheet" href="style.css">
    <link rel="manifest" href="manifest.json">
</head>
<body>
    <header>
        <h1>Moja Lista Zadań</h1>
        <nav>
            <a href="index.html" class="active">Lista</a>
            <a href="form.html">Dodaj Zadanie</a>
            <a href="api-data.html">Dane z API</a>
        </nav>
    </header>
    <main>
        <section id="todo-list-section">
            <h2>Zadania do zrobienia:</h2>
            <ul id="todo-list">
                <li class="placeholder">Brak zadań.</li>
            </ul>
        </section>

        <section id="add-task-shortcut" style="...">
             <h3>Szybkie dodawanie zadania:</h3>
            <form id="quick-add-form">
                <input type="text" id="quick-task-input" placeholder="Wpisz treść zadania..." required>
                <button type="submit">Dodaj szybko</button>
            </form>
        </section>
    </main>
    <footer>
        <p>&copy; 2025 Adam Zajler</p>
    </footer>

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


Moja Lista Zadań
Lista
Dodaj Zadanie
Dane z API
Zadanie dodane do listy!
Zadania do zrobienia:
Posprzątać pokój
Usuń
Szybkie dodawanie zadania:
Wpisz treść zadania...
Dodaj szybko
© 2025 Adam Zajler


Moja Lista Zadań
Lista
Dodaj Zadanie
Dane z API
Zadanie "Posprzątać pokój..." usunięte.
Zadania do zrobienia:
Brak zadań.
Szybkie dodawanie zadania:
Wpisz treść zadania...
Dodaj szybko
© 2025 Adam Zajler

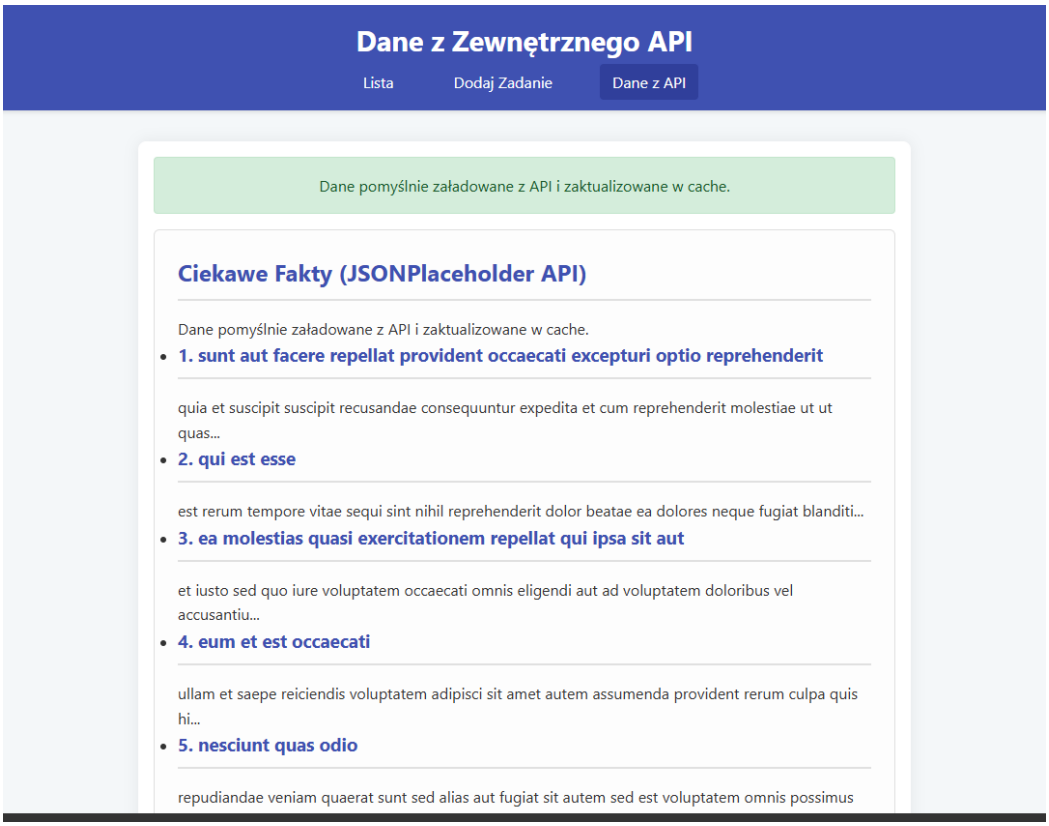
2) Dodawanie szczegółowych zadań (Formularz – form.html):
   a) Dedykowana podstrona z rozbudowanym formularzem pozwalającym na dodanie zadania z tytułem, opisem, terminem wykonania oraz priorytetem
   b) Prosta walidacja danych formularza (np. wymagany tytuł).
   c) Zapisywanie nowych zadań w bazie IndexedDB.

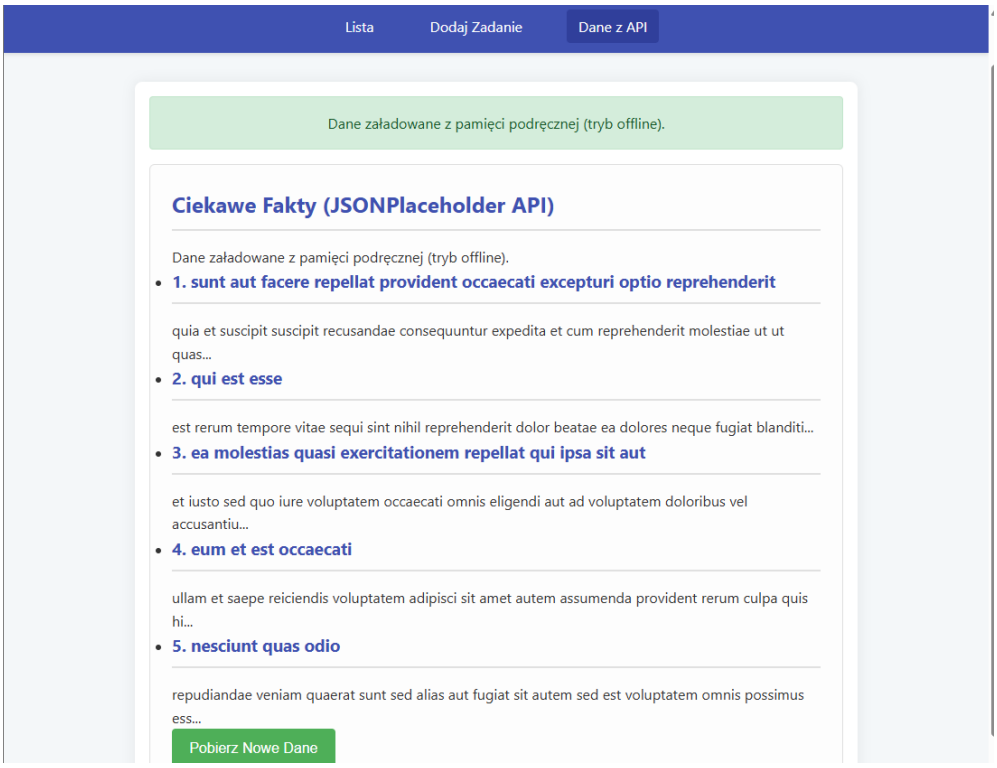




3) Wyświetlanie danych z zewnętrznego API (Dane z API – api.data.html):

a) Integracja z publicznym API (np. JSONPlaceholder) w celu pobrania i wyświetlenia przykładowych danych (np. listy postów).
b) Możliwość ręcznego odświeżenia danych z API.
c) Implementacja mechanizmu buforowania odpowiedzi API, umożliwiająca przeglądanie danych w trybie offline.





2.2 Działanie w trybie offline

Aplikacja została zaprojektowana z myślą o działaniu w trybie offline:

- Buforowanie zasobów aplikacji: Za pomocą Service Workera, kluczowe zasoby aplikacji (pliki HTML, CSS, JavaScript, manifest, ikony) są buforowane podczas pierwszej wizyty. Umożliwia to uruchomienie i nawigację po aplikacji bez dostępu do internetu.
- Lokalne przechowywanie danych: Zadania użytkownika są przechowywane w IndexedDB, co oznacza, że są dostępne i modyfikowalne nawet w trybie offline. Wszelkie zmiany są zapisywane lokalnie.
- Buforowanie danych z API: Dane pobierane z zewnętrznego API są również buforowane. Przy braku połączenia aplikacja próbuje załadować ostatnio pobrane dane z pamięci podręcznej.

3. Opis Techniczny Krok po Kroku

3.1. Konfiguracja środowiska

Do realizacji projektu wykorzystano następujące narzędzia i technologie:

- Edytor kodu: Webstorm
- Przeglądarka internetowa (testowa): Google Chrome (najnowsza wersja)
- Języki programowania: HTML5, CSS3, JavaScript (ES6+).
- System kontroli wersji: Git

3.2. Struktura projektu

Projekt został zorganizowany w następujący prosty sposób:

- index.html - Strona główna aplikacji
- form.html - Strona z formularzem dodawania zadań
- api-data.html - Strona wyświetlająca dane z API
- style.css - Główny arkusz stylów
- app.js - Główny plik JS aplikacji
- manifest.json - Plik manifestu PWA
- service-worker.js - Skrypt Service Workera
- js - Folder na dodatkowe skrypty JS
- form.js - Logika formularza z form.html i IndexedDB
- api.js - Logika pobierania i buforowania danych API

3.3. Tworzenie i podłączenie manifestu

Plik **manifest.json** dostarcza przeglądarce metadanych o aplikacji, umożliwiając jej "instalację" i kontrolę nad sposobem wyświetlania.

Zawartość przykładowego pliku **manifest.json**:

manifest.json

```
{
  "name": "Prosta Lista Zadań PWA",
  "short_name": "Lista Zadań",
  "description": "Prosta aplikacja PWA do zarządzania listą zadań.",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#3f51b5",
  "orientation": "portrait-primary"
}
```

Podłączenie do HTML: Manifest jest podłączany do każdego pliku HTML w sekcji **<head>**:

index.html

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

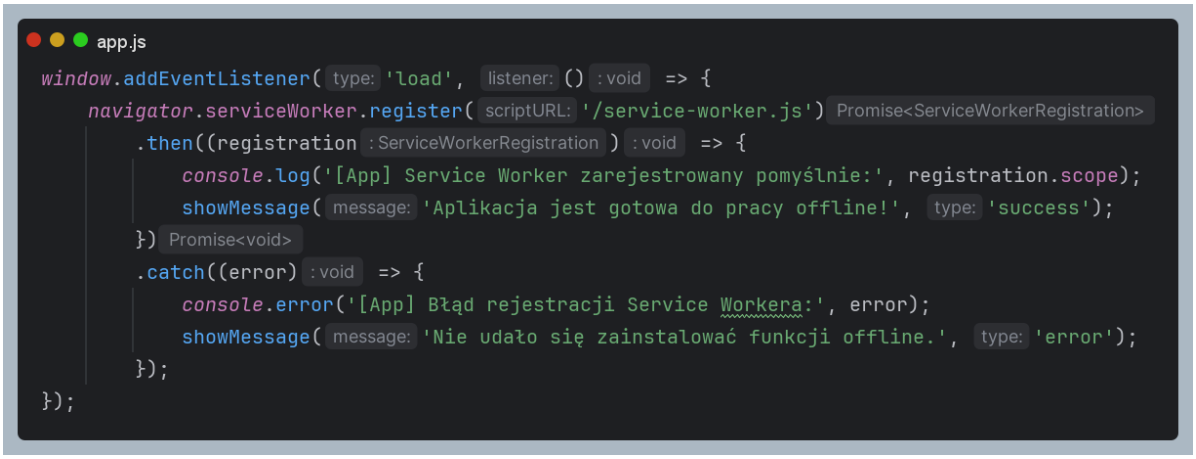
3.4. Tworzenie i rejestracja Service Workera

Service Worker (SW) to skrypt działający w tle, niezależnie od strony internetowej, umożliwiający m.in. przechwytywanie żądań sieciowych, zarządzanie pamięcią podręczną i obsługę powiadomień push.

Kluczowe elementy implementacji SW:

- Instalacja: Podczas instalacji SW, zdefiniowana lista zasobów (np. HTML, CSS, JS, ikony) jest pobierana i zapisywana w pamięci podręcznej (Cache API). Użyto **self.skipWaiting()** aby nowy SW aktywował się szybciej.

- Aktywacja: Podczas aktywacji, SW usuwa stare, nieużywane wersje pamięci podręcznej i przejmuje kontrolę nad klientami (self.clients.claim()).
- Przechwytywanie żądań (fetch): SW przechwytuje wszystkie żądania sieciowe pochodzące z aplikacji. Dla zasobów zdefiniowanych w **urlsToCache** zastosowano strategię *Cache First*: najpierw sprawdzana jest pamięć podręczna; jeśli zasób tam jest, jest zwracany. W przeciwnym razie, zasób jest pobierany z sieci, a następnie dodawany do cache.
- Rejestracja Service Workera (w `app.js`):



3.5. Integracja z API

Strona api-data.html integruje się z zewnętrznym API (dummyjson.com/todos) w celu pobrania i wyświetlenia danych.

- Pobieranie danych: Użyto standardowego fetch API do wysyłania żądań GET.
- Wyświetlanie danych: Pobrane dane (zadania "todos") są dynamicznie renderowane na stronie. Każde zadanie wyświetla swój tytuł, status (ukończone/do zrobienia) oraz ID użytkownika.
- Buforowanie odpowiedzi API: Zaimplementowano własną logikę buforowania w js/api.js:
    - Odpowiedzi z API są zapisywane w dedykowanej pamięci podręcznej (api-dummy-todos-cache-v1) przy użyciu Cache API.
    - Przy braku połączenia lub błędzie sieci, aplikacja próbuje załadować dane z tej pamięci podręcznej.
    - Przycisk "Pobierz Nowe Dane" pozwala na wymuszenie próby pobrania świeżych danych z sieci.

3.6. Aplikacja w trybie offline (Cache i Offline)

Żeby aplikacja działała szybko i była dostępna nawet bez połączenia z internetem, zastosowano różne sposoby zapisywania danych w pamięci podręcznej przeglądarki.

- Główne pliki aplikacji: Tutaj działa zasada Cache First. Plik service-worker.js dba o to, żeby najważniejsze pliki (HTML, CSS, JavaScript) zostały zapisane w pamięci podręcznej już podczas pierwszej instalacji aplikacji. Gdy użytkownik ponownie otwiera aplikację, pliki te są ładowane od razu z cache, co przyspiesza start. Jeśli jakiegoś pliku nie ma w cache (lub Service Worker jest aktualizowany), dopiero wtedy jest on pobierany z Internetu. Service Worker dba również o to, by nowe wersje plików były dodawane do cache przy aktualizacji.
- Dane pobierane z internetu (API): Dla danych, które często się zmieniają (jak te z dummyjson.com), zastosowano zasadę Network first, then cache. Jest to opisane w pliku js/api.js. Aplikacja najpierw próbuje pobrać świeże dane. Jeśli się uda, zapisuje je do specjalnej pamięci podręcznej i wyświetla. Jeśli internet nie działa lub API ma problemy, aplikacja sięga po ostatnio zapisaną kopię z cache.
- Dane użytkownika: Wszystkie zadania, które zostaną dodane do listy, są zapisywane w lokalnej bazie danych przeglądarki zwanej IndexedDB. Dzięki temu są one zawsze dostępne, nawet gdy użytkownik jest offline.

3.7. Gdzie umieścić aplikację w internecie (Hosting)

4. Wyzwania i rozwiązania

Podczas tworzenia aplikacji napotkano na kilka typowych wyzwań.

Problem: Odświeżanie Service Workera

Charakterystyka problemu: Zauważono, że po wprowadzeniu zmian w kodzie Service Workera lub powiązanych plikach, użytkownik nadal mógł widzieć starą wersję aplikacji. Service Worker nie zawsze odświeżał się od razu.

Rozwiązanie:

- Za każdym razem, gdy wprowadzono znaczące zmiany, zmieniano nazwę pamięci podręcznej w pliku service-worker.js. Dzięki temu stary cache był automatycznie usuwany.

- W celu przyspieszenia przejmowania kontroli przez nowego Service Workera, zastosowano specjalne komendy (self.skipWaiting() i self.clients.claim()).

Wnioski: Aby użytkownicy zawsze mieli dostęp do najnowszej wersji aplikacji, należy dobrze rozumieć mechanizmy działania Service Workera i starannie planować proces aktualizacji oraz używać odpowiednich narzędzi developerskich.

Problem: Praca z bazą danych IndexedDB

Charakterystyka problemu: Zapisywanie i odczytywanie danych z IndexedDB jest operacją asynchroniczną, co początkowo mogło wydawać się skomplikowane, zwłaszcza przy obsłudze wielu zadań.

Rozwiązanie:

- Kod odpowiedzialny za obsługę IndexedDB podzielono na mniejsze, bardziej zrozumiałe funkcje.
- Wdrożono dokładną obsługę wszystkich możliwych odpowiedzi i błędów pochodzących z IndexedDB.
- Korzystano z narzędzi deweloperskich w celu podglądu stanu bazy danych.

Wnioski: Praca z IndexedDB wymaga cierpliwości, ale zastosowanie Promises znacznie porządkuje kod.

Problem: Strategia zapisu danych z zewnętrznego API

Charakterystyka problemu: Proste, "sztywne" zapisywanie danych z API, podobne do cachowania plików aplikacji, okazało się nieodpowiednie ze względu na dynamiczny charakter tych danych.

Rozwiązanie:

- W pliku js/api.js stworzono system, który najpierw próbuje pobrać dane z internetu.
- W przypadku udanego pobrania, dane są zapisywane do specjalnej pamięci podręcznej i prezentowane użytkownikowi.
- Jeśli połączenie z internetem jest niedostępne, aplikacja wczytuje ostatnio zapisaną wersję danych.

- Dodano przycisk "Pobierz Nowe Dane", umożliwiający użytkownikowi ręczne odświeżenie informacji.

Wnioski: Różne typy danych wymagają różnych strategii cachowania. Ważne jest również informowanie użytkownika o źródle danych (np. "wczytano z pamięci").

5. Podsumowanie

5.1. Co działa dobrze?

- Aplikacja jest "prawdziwą" PWA: można ją zainstalować, działa bez internetu, ma plik manifestu i Service Workera.
- Główne funkcje listy zadań (dodawanie, usuwanie, oznaczanie jako zrobione) działają bez zarzutu i korzystają z lokalnej bazy IndexedDB.
- Pobieranie danych z zewnętrznego API i ich zapisywanie w pamięci podręcznej działa tak, jak powinno, dzięki czemu można z nich korzystać offline.
- Można płynnie przechodzić między trzema głównymi ekranami aplikacji.
- Wygląd jest prosty i dostosowuje się do różnych ekranów.

5.2. Co można by jeszcze ulepszyć?

- Informowanie o błędach: Można by lepiej informować użytkownika, co poszło nie tak i co może zrobić.
- Wygląd i obsługa (UI/UX): Aplikacja jest prosta wizualnie. Można by ją upiększyć, dodać animacje i ogólnie poprawić komfort użytkowania.
- Synchronizacja danych w tle: Na razie dane są tylko lokalnie. W przyszłości można by dodać opcję synchronizacji z serwerem, gdy pojawi się internet.
- Szybkość działania: Przy bardzo dużej liczbie zadań, operacje na bazie danych i wyświetlanie listy mogłyby być jeszcze szybsze.
- Testy: Nie mamy automatycznych testów. W większym projekcie byłyby konieczne.

5.3. Czego ważnego się nauczyłem?

Tworzenie tej aplikacji od zera, bez użycia gotowych frameworków, było ciekawą lekcją; jednakże było to trudniejsze niż z użyciem frameworków.
Najważniejsze rzeczy, których się nauczyłem, to:

- Jak w praktyce używać Service Workerów: jak działają, jak zapisywać dzięki nim pliki i jak przechwytywać żądania do internetu.
- Do czego służy Plik Manifestu Aplikacji i jak go skonfigurować.
- Jak pracować z bazą IndexedDB do przechowywania danych w przeglądarce, w tym jak radzić sobie z operacjami, które nie wykonują się od razu.
- Jak sprawić, by aplikacja działała bez internetu.
- Jak rozwiązywać typowe problemy przy tworzeniu PWA, np. z aktualizacją Service Workera.
- Jak wiele można zrobić używając "czystego" JavaScriptu i jak działają podstawowe mechanizmy przeglądarki.

6. Załączniki

Link do aplikacji: https://adamzajler.github.io/todo-app-pwa/

Link do repozytorium: https://github.com/AdamZajler/todo-app-pwa

* **Link do działającej aplikacji:** [W tym miejscu umieść link do hostowanej aplikacji]

* **Link do repozytorium GitHub z kodem źródłowym:** [W tym miejscu umieść link do repozytorium GitHub]

* **Zrzuty ekranu pokazujące aplikację w trybie offline i online:**

  * [Miejsce na zrzut ekranu - tryb online]

  * [Miejsce na zrzut ekranu - tryb offline]

  * [Miejsce na zrzut ekranu - lista zadań]

  * [Miejsce na zrzut ekranu - formularz]

  * [Miejsce na zrzut ekranu - dane z API]

* **Screenshoty z Lighthouse (wynik PWA >90 punktów wymagany):**

  * [Miejsce na zrzut ekranu z audytu Lighthouse - kategoria PWA]