Sprawozdanie z laboratorium 3

Mikołaj Kubś 272662

30 kwietnia 2025

1 Cel zadania

Celem zadania jest stworzenie prostej aplikacja PWA z wykorzystaniem HTML, CSS i JS. Kolejnym krokiem był deployment i testy funkcjonalności.

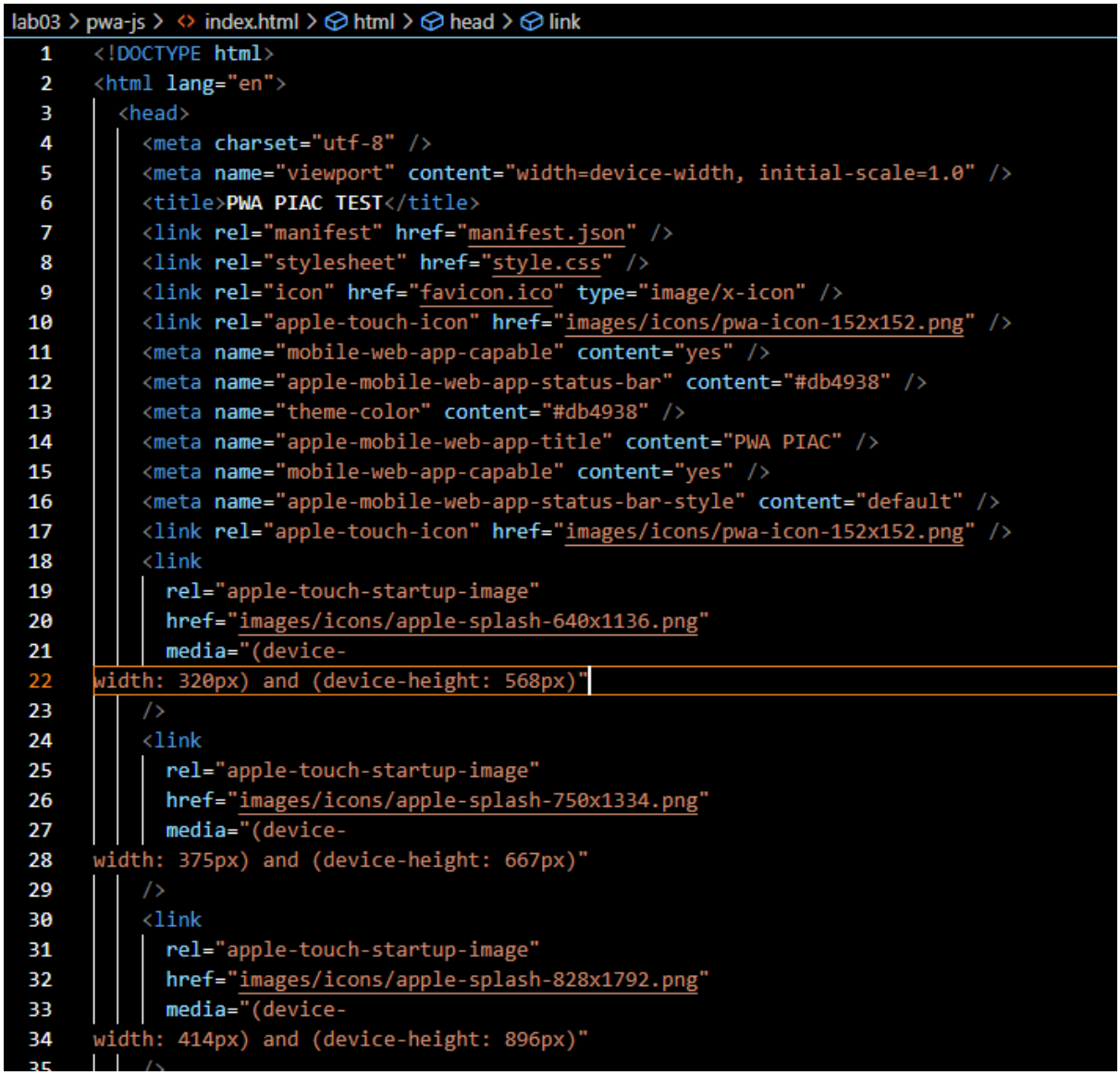
1.1 Wprowadzenie

Progressive Web App to aplikacja internetowa uruchamiana tak jak zwykła strona internetowa, ale umożliwiająca stworzenie wrażenia działania jak natywna aplikacja mobilna (lub aplikacja desktopowa).

1.2 Tworzenie aplikacja PWA z wykorzystaniem HTML, JS i CSS

1.2.1 Kod HTML

Plik `index.html` definiuje prosty kod HTML aplikacji wraz z ikonami dla PWA i innymi informacjami:



Rysunek 1: Kod index.html

1.2.2 Stylizacja CSS

Plik `style.css` definiuje kaskadowe arkusze stylów:

lab03 > pwa-js > style.css > html

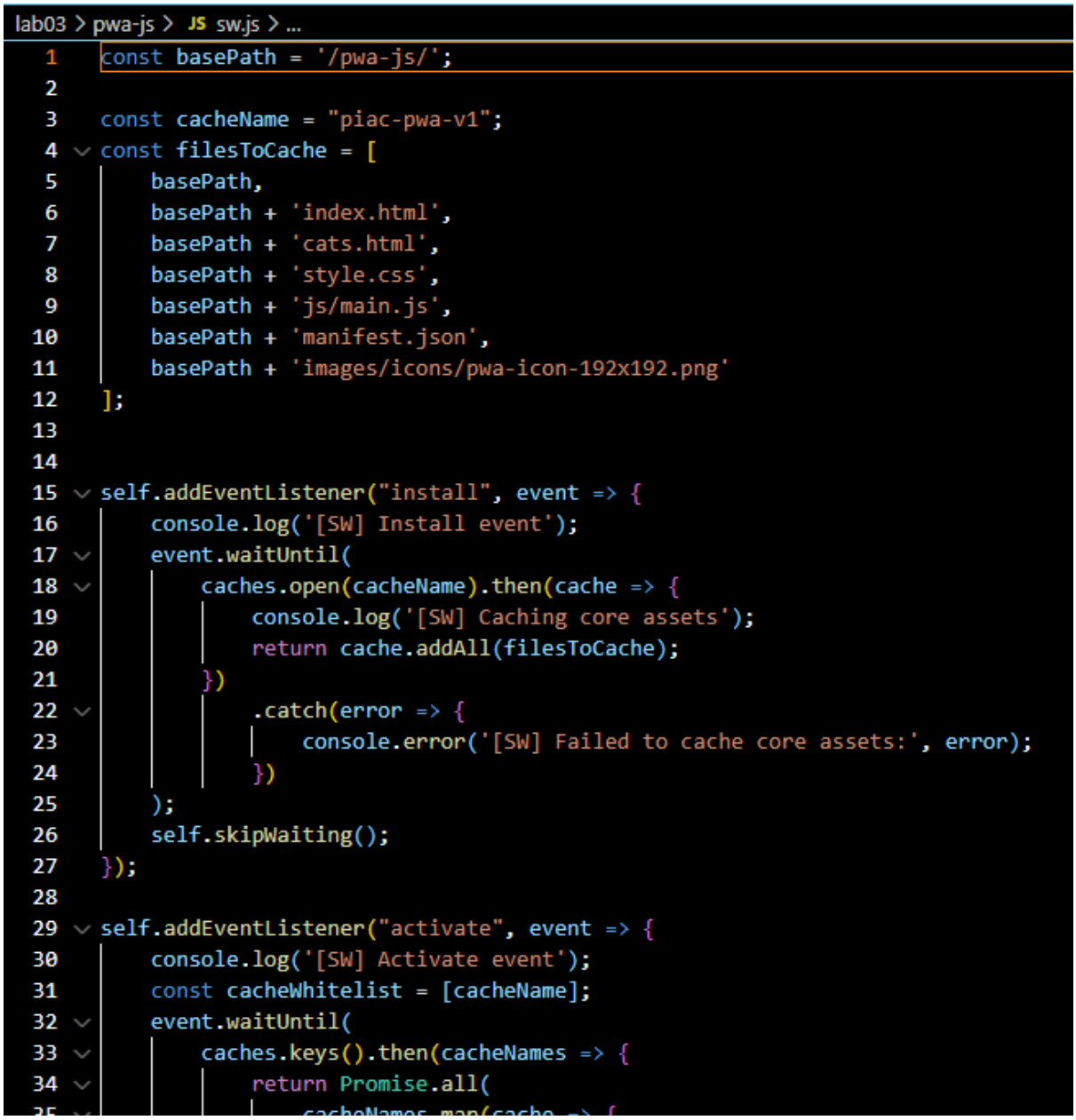
```css
html,
body {
  font-family: sans-serif;
  text-align: center;
  height: 100%;
  margin: 0;
  padding: 0;
  width: 100%;
}
.container {
  margin: auto;
  text-align: center;
}
main {
  padding: 2rem;
  font-size: 1rem;
}
.title {
  font-size: 3rem;
  animation: fadeIn 2s ease-in-out;
}
@keyframes fadeIn {
  from {
    opacity: 0;
    transform: translateY(-20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
.dog-image {
  width: 200px;
  height: 200px;
```

1.2.3 Kod JavaScript

lab03 > pwa-js > js > JS main.js > ...

```javascript
import { initializeApp } from 'https://www.gstatic.com/firebasejs/10.6.0/firebase-app.js';
import { getMessaging, getToken, onMessage } from 'https://www.gstatic.com/firebasejs/10.6.0/firebase-messaging.js';

const firebaseConfig = {
    apiKey: "AIzaSyAWlxciG0vZSLD78nC2n2qGNOvuzul3rTE",
    authDomain: "pwa-ts-c7c15.firebaseapp.com",
    projectId: "pwa-ts-c7c15",
    storageBucket: "pwa-ts-c7c15.firebasestorage.app",
    messagingSenderId: "438732315703",
    appId: "1:438732315703:web:bcc4ff82c8ce5e4b1f73b2",
    measurementId: "G-7M93B09YP4"
};

const app = initializeApp(firebaseConfig);
const messaging = getMessaging(app);

const VAPID_KEY = "BDSUjXiUoetn8Qz6mmTlIEFL9Mv6Z-CCmuFofvAk1Uit5mn2TV4EMNHze1ei90PA6maIhR-e563M8XD5ffyiItY";
const MESSAGING_SW_PATH = '/pwa-js/firebase-messaging-sw.js';
const MESSAGING_SW_SCOPE = '/pwa-js/';
const MAIN_SW_PATH = '/pwa-js/sw.js';
const MAIN_SW_SCOPE = '/pwa-js/';

let firebaseMessagingSwReg = null;

if ("serviceWorker" in navigator) {
    window.addEventListener("load", () => {
        navigator.serviceWorker
            .register(MAIN_SW_PATH, { scope: MAIN_SW_SCOPE })
            .then(registration => {
                console.log("Main PWA Service Worker registered:", registration);
            })
            .catch(error => {
                console.error("Main PWA Service Worker registration failed:", error);
            });
```

Rysunek 3: Fragment kodu main.js



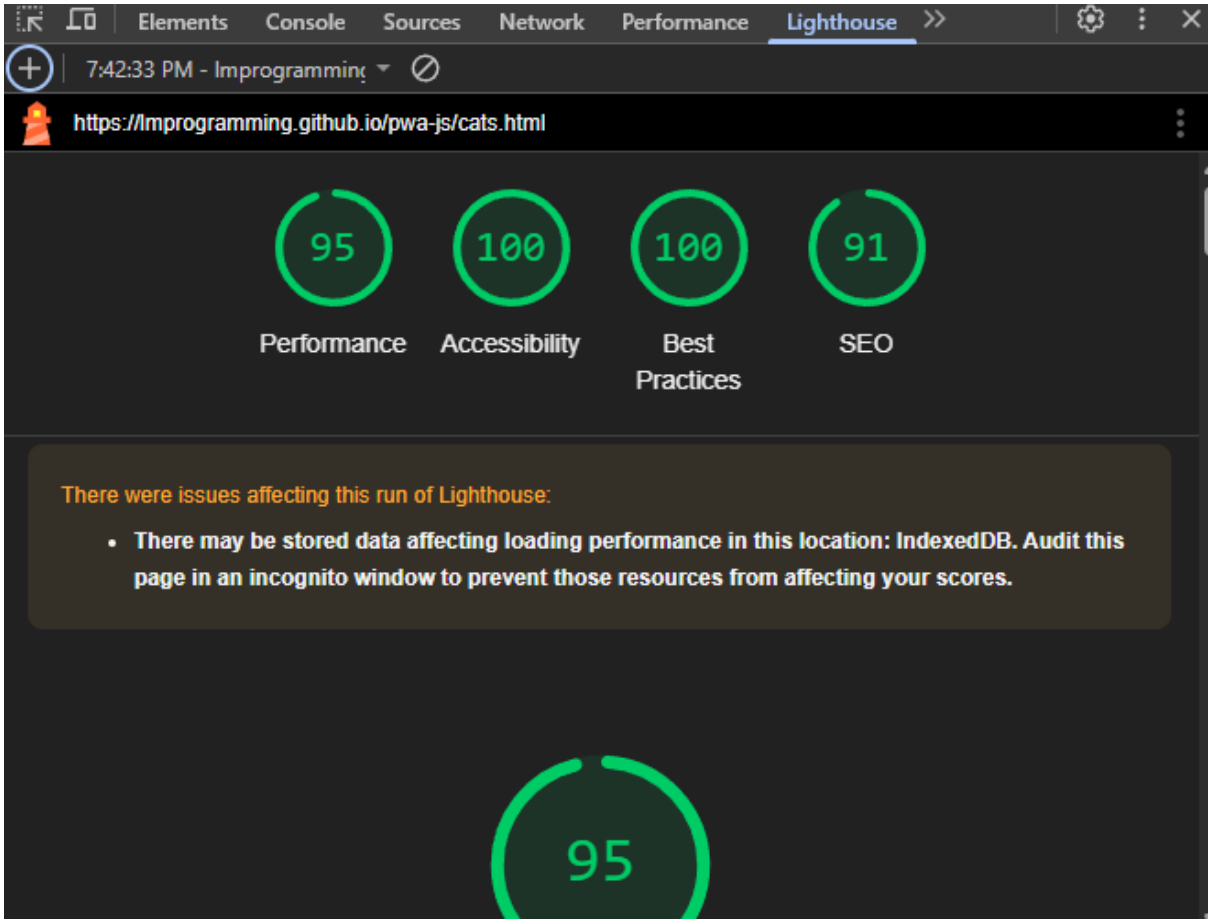
Rysunek 4: Fragment kodu sw.js - service worker

1.3 Dalsza implementacja

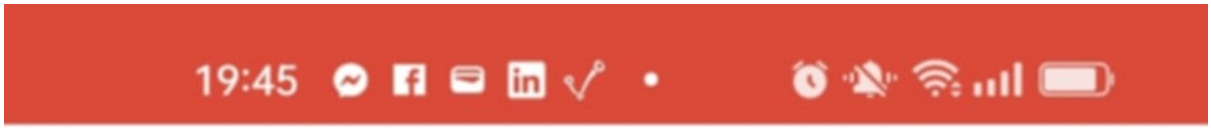
Dzięki wykorzystaniu kodu w instrukcji implementacja przebiegła bez większych problemów. Najwięcej problemów wynikło ze ścieżek, gdyż publikując stronę w GitHub Pages okazuje się, że jest głębiej w folderze.

Do ikon wykorzystano oba podane narzędzia: pwa-asset-generator i RealFaviconGenerator.

Rozbudowano stronę, dodając zdjęcia psów i podstronę ze zdjęciami kotów.



Rysunek 5: Wynik analizy Lighthouse

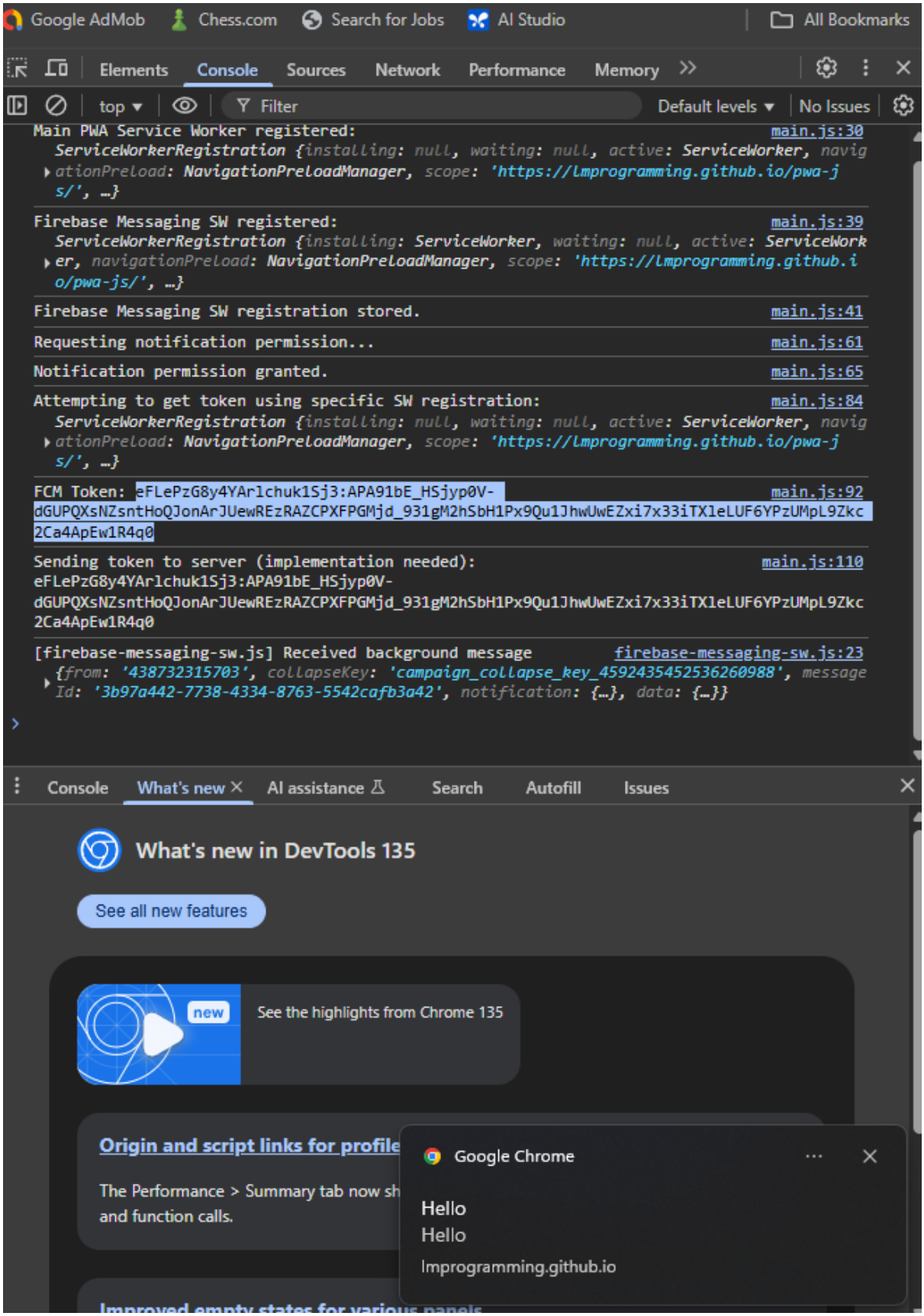

19:45

Service worker działa jak należy, cache'ując odpowiednie pliki. Po pobraniu aplikacji na telefon splash image się pokazuje.

Wdrożono aplikację w GitHub pages, co, jak wyżej wspomniano, wymagało naprawy ścieżek.

1.4 Powiadomienia

Sprawiły one najwięcej problemów implementacyjnych. Stworzono projekt w Firebase i skrypt `firebase-messaging-sw.js`. Po poprawie ścieżek w końcu zadziałało i dzięki wydrukowaniu tokenu FCM można było wysłać testową wiadomość z konsole Firebase.



Rysunek 7: Powiadomienie