

PWA

Pierwsze kroki

Stwórz:

1. Folder `css` a w nim plik `style.css`
2. Folder `js` a w nim plik `main.js`
3. Folder `images`
4. W głównym folderze plik `index.html`
5. Utwórz repozytorium w serwisie `github` i stwórz dla niego `github pages`.

Interface

Podczas pisania PWA należy pamiętać o dwóch wymaganiach:

- Aplikacja powinna wyświetlać część treści, nawet jeśli JavaScript jest wyłączony. Zapobiega to wyświetlaniu pustej strony, jeśli połączenie internetowe jest słabe lub używa się starszej przeglądarki.
- Aplikacja powinna reagować i wyświetlać się poprawnie na różnych urządzeniach. Innymi słowy, musi być przyjazny dla urządzeń mobilnych.

Kod HTML:

```
<!doctype html>
<html lang="en">

<head>
<meta charset="utf-8">
<title>Aplikacja - studia podyplomowe</title>
<link rel="stylesheet" href="./css/style.css">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body class="fullscreen">
<div class="container">
    <h1 class="title">Studia podyplomowe</h1>
    <h2 class="title">PWA</h2>
</div>
</body>

</html>
```

Kod CSS:

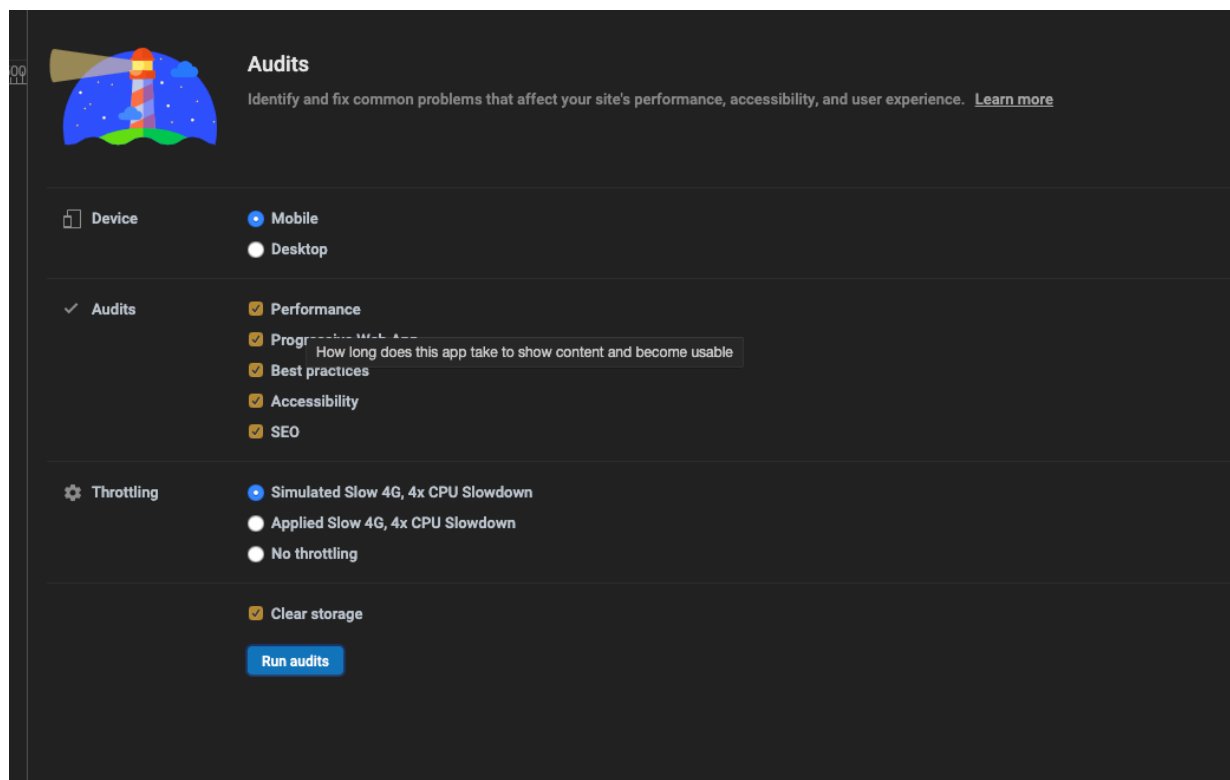
```
html,
.fullscreen {
display: flex;
height: 100%;
width: 100%;
}

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

Testowanie aplikacji

Wyślij zmiany do zdalnego repozytorium, Wyświetl stronę i sprawdź, czy wszystko jest w porządku!

Teraz, gdy mamy stronę w przeglądarce, użyjemy Google Lighthouse do przetestowania aplikacji i sprawdzenia, czy jest zgodna ze standardami PWA. Naciśnij klawisz F12, aby otworzyć panel programisty w Chrome, a następnie kliknij kartę audytu, aby otworzyć Lighthouse.



Upewnij się, że opcja „Progressive Web App” jest zaznaczona. Możesz na razie odznaczyć inne. Następnie kliknij przycisk „Run audits”. Po minucie lub dwóch `Lighthouse` dać Ci wynik i listę audytów, które aplikacja przeszła lub nie zdała.

Aplikacja powinna w tym momencie zdobyć około 45 punktów. Jeśli wszystko zostało poprawnie zakodowane, zauważysz, że większość testów, które przejdzie, jest związana z wymaganiami, które przedstawiliśmy na początku:

Service worker

Kolejnym wymaganiem dla naszej aplikacji jest zarejestrowanie `service worker` - są to skrypty działające w tle, wykonujące zadania niewymagające interakcji z użytkownikiem.

W naszej aplikacji użyjemy jednego do pobrania i buforowania naszej zawartości, a następnie dostarczenia jej z pamięci podręcznej, gdy użytkownik będzie offline.

Utwórz plik o nazwie `sw.js` w folderze głównym i wprowadź treść skryptu poniżej. Powodem, dla którego jest zapisany w katalogu głównym aplikacji, jest zapewnienie dostępu do wszystkich plików aplikacji. Dzieje się tak, ponieważ `service workers` mają jedynie uprawnienia dostępu do plików w tym samym katalogu i podkatalogach.

Kod w pliku `sw.js` :

```

var cacheName = "podyplomowe";
var filesToCache = ["/", "./index.html", "./css/style.css", "./js/main.js"];

/* Start the service worker and cache all of the app's content */
self.addEventListener("install", function (event) {
  // Perform install steps
  event.waitUntil(
    caches.open(cacheName).then(function (cache) {
      console.log("Opened cache");
      return cache.addAll(filesToCache);
    })
  );
});

/* Serve cached content when offline */
self.addEventListener("fetch", function (event) {
  event.respondWith(
    caches.match(event.request).then(function (response) {
      // Cache hit - return response
      if (response) {
        return response;
      }
      return fetch(event.request);
    })
  );
});

```

Pierwsze wiersze skryptu deklarują dwie zmienne:

- `cacheName`
- `filesToCache` .

`cacheName` służy do tworzenia pamięci podręcznej offline w przeglądarce i daje nam do niej dostęp z poziomu Javascript. `filesToCache` to tablica zawierająca listę wszystkich plików, które należy buforować. Pliki te powinny być zapisane w formie adresów URL. Zauważ, że pierwszy to po prostu „/”, podstawowy adres URL. Dzieje się tak, dlatego przeglądarka buforuje `index.html` , nawet jeśli użytkownik nie wpisuje bezpośrednio nazwy tego pliku.

Następnie dodajemy funkcję, aby zainstalować proces roboczy usługi i utworzyć pamięć podręczną przeglądarki przy użyciu `cacheName` . Po utworzeniu pamięci podręcznej dodaje wszystkie pliki wymienione w tablicy `filesToCache` .

Na koniec dodajemy funkcję ładowania plików buforowanych, gdy przeglądarka jest w trybie offline. Po utworzeniu skryptu procesu roboczego usługi musimy go zarejestrować w naszej

aplikacji. Utwórz plik o nazwie `main.js` w folderze `js` i wprowadź następujący kod:

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

  if ('serviceWorker' in navigator) {
    navigator.serviceWorker
      .register('./sw.js');
  }
}
```

Ten kod po prostu ładuje skrypt procesu roboczego usługi i uruchamia go.

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

Dodaj kod do swojej aplikacji, dołączając skrypt tuż przed tagiem zamykającym w `index.html`.

Wyślij zmiany do zdalnego repozytorium, sprawdź czy nie ma błędów i ponownie uruchom audyt.

Manifest

Ostatnim wymaganiem dla PWA jest posiadanie pliku manifestu. Manifest to plik json, który służy do określania wyglądu i zachowania aplikacji na urządzeniach. Na przykład możesz ustawić orientację aplikacji i kolor motywu.

Zapisz plik o nazwie `manifest.json` w folderze głównym i dodaj następującą treść:

```
{
  "name": "podyplomowe",
  "short_name": "podyplomowe",
  "lang": "en-US",
  "start_url": "./index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}
```

Dla naszej aplikacji ustalamy tytuł, kolory tła i motywu oraz informujemy przeglądarkę, że powinna być traktowana jako samodzielna aplikacja bez przeglądarki Chrome.

Wiersz po wierszu pola są następujące: **name** Tytuł aplikacji. Służy to do monitorowania użytkownika o zainstalowanie aplikacji. Powinien to być pełny tytuł aplikacji. **short_name**

Jest to nazwa aplikacji, która nie pojawi się na ikonie aplikacji. To powinno być krótkie i na temat. **lang** Domyślny język, w którym zlokalizowana jest aplikacja. W naszym przypadku angielski. **start_url** Informuje przeglądarkę, którą stronę załadować po uruchomieniu aplikacji. Zwykle będzie to index.html, ale nie musi tak być. **display** Typ powłoki, w której powinna pojawić się aplikacja. W naszej aplikacji używamy autonomicznej aplikacji, aby wyglądała jak standardowa aplikacja natywna. Istnieją inne ustawienia, aby ustawić pełny ekran lub włączyć przeglądarkę Chrome. **background_color** Kolor ekranu powitalnego, który otwiera się po uruchomieniu aplikacji. **theme_color** Ustawia kolor paska narzędzi i przełącznika zadań.

Aby dodać manifest do aplikacji, połącz go z nim w tagu `head` w pliku `index.html` w następujący sposób:

```
<link rel="manifest" crossorigin="use-credentials" href="./manifest.js
on" />
```

Powinieneś także zadeklarować kolor motywu, aby pasował do zestawu ustawionego w manifestcie, dodając metatag wewnątrz sekcji `head` :

```
<meta name="theme-color" content="white" />
```

Wyślij zmiany do zdalnego repozytorium, sprawdź czy nie ma błędów i ponownie uruchom audyt.

App Icons

Po poprzednim kroku możesz zauważyć, że Lighthouse narzeka na brak ikon aplikacji. Chociaż nie jest to absolutnie konieczne, aby aplikacja działała w trybie offline, pozwalają użytkownikom dodawać aplikację do ekranu głównego.

Aby poprawnie dodać tę funkcję, potrzebujesz ikony aplikacji dostosowanej do przeglądarki, Windowsa, Maca / iPhone'a i Androida. To minimum 7 różnych rozmiarów:

- 128 x 128 pikseli,
- 144 x 144 pikseli,
- 152 x 152 pikseli,
- 192 x 192 pikseli,
- 256 x 256 pikseli,
- 512 x 512 pikseli
- favicon 16 x 16 pikseli.

Dodaj ikony do pliku manifestu po właściwości `short_name` w następujący sposób:

```

{
  "name": "podyplomowe",
  "short_name": "podyplomowe",
  "icons": [
    {
      "src": "./images/hello-icon-128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "./images/hello-icon-144.png",
      "sizes": "144x144",
      "type": "image/png"
    },
    {
      "src": "./images/hello-icon-152.png",
      "sizes": "152x152",
      "type": "image/png"
    },
    {
      "src": "./images/hello-icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "./images/hello-icon-256.png",
      "sizes": "256x256",
      "type": "image/png"
    },
    {
      "src": "./images/hello-icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "lang": "en-US",
  "start_url": "./index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}

```

Dodaj pozostałe tagi do sekcji `head` w pliku `index.html` :

```
<!doctype html>
<html lang="en">

<head>
<meta charset="utf-8">
<title>Hello World</title>
<link rel="manifest" crossorigin="use-credentials" href="./manifest.js
on" />
<link rel="stylesheet" href="./css/style.css">
<link rel="icon" href="./favicon.ico" type="image/x-icon" />
<link rel="apple-touch-icon" href="images/hello-icon-152.png">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="theme-color" content="white" />
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="apple-mobile-web-app-title" content="Hello World">
<meta name="msapplication-TileImage" content="./images/hello-icon-144.
png">
<meta name="msapplication-TileColor" content="#FFFFFF">
</head>

<body class="fullscreen">
  <div class="container">
    <h1 class="title">Studia podyplomowe</h1>
    <h2 class="title">PWA</h2>
  </div>
<script src="./js/main.js"></script>
</body>




</html>
```

Wyślij zmiany do zdalnego repozytorium, sprawdź czy nie ma błędów i ponownie uruchom audyt.

Instalowanie PWA

W `ChromeDev Tools` przejdź do zakładki `Application` i sprawdź, czy nie występują żadne błędy.

Jezeli nie stwierdzono zadnych błędów, to powinienes zobaczyć:

  <https://marcindl.github.io/PWA-sample-01/> 

Zainstaluj aplikację i mozesz juz z niej korzystać!

Źródła:

- <https://developers.google.com/web/fundamentals/primers/service-workers>
- https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API
- https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Installable_PWAs
- https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Add_to_home_screen#Manifest