Ćwiczenie 6

1. Cel ćwiczenia

W języku HTML5 można tworzyć nie tylko witryny lecz także aplikacje bazujące na języku HTML5 (np. aplikacje biurowe). Aplikacje jako takie powinny mieć możliwość działania także w przypadku gdy nie posiadamy dostępu do sieci (np. jesteśmy w podróży). Celem ćwiczenia będzie wykorzystanie technik do tworzenia aplikacji w trybie odłączenia (offline).

2. Teoria.

HTML5 nie jest już językiem stricte dokumentów WWW. Obecnie programiści wykorzystują go między innymi do:
- tworzenia aplikacji internetowych działających niczym tradycyjne aplikacje;dzięki temu programy zyskują pożądaną niezależność od platformy systemowej, a nawet sprzętowej (dba o to producent przeglądarki)
- tworzenia graficznych interfejsów aplikacji systemowych; dzięki temu zabiegowi inżynierowie oprogramowania mogą tworzyć pożądaną funkcjonalność aplikacji, pozostawiając projekt graficzny projektantom znającym najczęściej właśnie HTML. Ponadto zabieg ten uniezależnia część wizualną programu od jego części logicznej – dzięki temu zmiany dokonywane w jednej z nich nie mają bezpośredniego przełożenia na drugą z nich (innymi słowy aplikacje powinny działać stabilniej)
- tworzenie animacji i gier – zarówno w przeglądarce jak i jako osobnych tytułów. Dzięki implementacji płótna oraz WebGL pisanie stron i aplikacji multimedialnych jest znacznie prostsze. Całość opiera się bowiem o język JavaScript.
- tworzenie interfejsów systemów operacyjnych – obecnie na rynku istnieje kilka systemów operacyjnych, których interfejs użytkownika napisany jest przy użyciu HTML5 bądź XHTML5 (ze względu na rozszerzalność atrybutów).

Sytuacja ta wymagała dodania możliwości pracowania aplikacji bez dostępu do sieci (tzw. offline). Wyobraźmy sobie bowiem np. księgową, która nie może wystawić faktury bo z przyczyn niezależnych utraciła dostęp do sieci internet. Bojąc się tego typu incydentu, bez odpowiedniego zabezpieczenia zapewne nigdy nie wybrałaby wygodnego, aczkolwiek obarczonego tego typu ryzykiem programu do księgowości.

Stąd w pierwszej specyfikacji HTML5 pojawił się opis Application Cache (pamięć podręczna aplikacji). Rozwiązanie to pozwala na wskazanie przeglądarce które pliki powinna bezwzględnie zapamiętać lokalnie i pozwalać na otwieranie ich nawet jeżeli brak jest połączenia z internetem. Implementację tego rozwiązania posiadają wszystkie przeglądarki - zarówno komputerowe jak i przenośne (np. na tablecie). Niestety w 3 kwartale 2016 roku technologia została oficjalnie **WYRZUCONA ZE STANDARDU** – chodzi o względy bezpieczeństwa (wtryskiwanie niebezpiecznego kodu). Oznacza to, że obsługa tego rozwiązania może oficjalnie zostać porzucona i nie należy wykorzystywać tego rozwiązania w nowych projektach. Trzeaba jednak pamiętać, że aplikacje wykorzystujące to rozwiązanie będą mogły jeszcze jakiś czas działać – większość dostawców przeglądarek wyrzuca jedynie odpowiedni komunikat w konsoli JavaScript.

Obecnie tworzenie aplikacji tzw. offline możliwe jest dzięki Service Workers. Rozwiązanie to nie jest jeszcze OFICJALNYM standardem. Zaproponowane zostało przez projektantów Chromium/Chrome i obecnie wdrożone je dla Chrome/Chromium, Firefox, Edge oraz Opera. Swojej implementacji nie posiada Internet Explorer (i nie będzie posiadał – przeglądarka nie jest rozwijana) oraz Safari.

Samo rozwiązanie bazuje bezpośrednio na obiekcie Promise, którego głównym zastosowaniem jest asynchroniczne przetwarzanie danych i skryptów JavaScript (edycja 6 standardu ECMAScript). Najprościej rzecz ujmując odpowiedni skrypt (napisany przez nas) ma za

zadanie w tle dodać do pamięci podręcznej pliki strony WWW którą użytkownik obecnie przegląda. W przypadku, gdy użytkownik będzie chciał wrócić na stronę po utracie połączenia, ten sam skrypt ma za zadanie wybrać wskazane pliki z pamięci przeglądarki i otworzyć adres tak, jakby pliki tyle co zostały pobrane z serwera.

UWAGA! Rozwiązanie to nie będzie działać w przypadku użytkowania trybu prywatnego. Będzie bezużyteczne w przypadku wyłączenia przez użytkownika pamięci podręcznej przeglądarki. Dodatkowo nie zadziała jeżeli użytkownik opróżni pamięć podręczną przeglądarki (ręcznie bądź przy pomocy narzędzia porządkującego dysk twardy).

UWAGA! Service Workers działa JEDYNIE z protokołem HTTPS. W przypadku HTTP obiekt nie zostanie utworzony/strona nie zostanie załadowana do pamięci podręcznej. Protokół HTTP działa jedynie w przypadku adresów localhost/127.0.0.1 (bądź jeżeli użyjemy flag i opcji przeglądarek do ignorowania bezpieczeństwa – co nie jest polecane do codziennego użytku).

UWAGA! Poniższy materiał omawia (wraz z przykładem) jedynie działanie Service Workers. Zakłada się więc, że czytający zna działanie obiektu Promise. Jeżeli nie – dobrym pomysłem byłoby wstępne zapoznanie się z nim (odnośnik w materiałach).

2.1 Podstawy użytkowania Service Workers

Najważniejszą częścią tworzenia aplikacji jest załadowanie listy plików do zapamiętania oraz przestrzeni, w której będą one przechowywane. Ponieważ jednak nie wszystkie przeglądarki posiadają zaimplementowaną opisywaną technologię należy najpierw sprawdzić czy możemy z niej skorzystać:

```
if ('serviceWorker' in navigator) {
        //kod Service Worker
}
```

Jeżeli test przebiegnie negatywnie to nie zatrzyma to ładowana dalszej części naszej strony.

Rejestrowanie usługi (kod należy wstawić w miejsce komentarza z poprzedniego przykładu):

```
 navigator.serviceWorker.register('/cache-test/cc.js', {scope: '/cache-test/'})
  .then(function(reg) {
    // rejestracja przebiegła poprawnie
    console.log('Zarejestrowano w zakresie:  ' + reg.scope);
  }).catch(function(error) {
    // rejestracja przebiegła z błędem (cache nie działa)
    console.log('Błąd rejestracji -  ' + error);
  });
```

Powyższy kod zapamięta wszystko ze wskazanego zakresu (będącego wskazanym folderem). Oznacza to, że wszystkie pliki html/css/js/graficzne itp. wraz z podfolderami znajdujące się we wskazanym folderze będą przechowywane w pamięci podręcznej. Opcja dodatkowa, jaką jest cache, nie musi być podawana (domyślnie przyjęty zostanie katalog zawierający plik cc.js).

UWAGA! MAKSYMALNYM zakresem pamięci podręcznej jest lokalizacja pliku wątku (worker) wywoływanego przez funkcję register. Jeżeli będziemy chcieli przechować pliki z wyższej lokalizacji (np. folderu głównego strony) to będziemy musieli przenieść plik cc.js do tego katalogu.

Kolejnym etapem będzie stworzenie pliku cc.js we wskazanej lokalizacji. Plik będzie tzw. zarządcą pamięci podręcznej - będzie zawierał informacje o plikach i katalogach naszej strony, które mają być dostępne po rozłączeniu się z siecią. Dodatkowo będzie odpowiednio reagował w zależności czy jesteśmy podłączeni do sieci (porównamy/zaktualizujemy zawartość pamięci) czy nie (wyświetlimy zawartość aktualnie pobraną).

Zawartość pliku przedstawiona będzie w kilku częściach. Najpierw przyjrzyjmy się głównemu zdarzeniu odpowiedzialnemu za zapisywanie odpowiednich części naszej strony w pamięci podręcznej:

```
this.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open('version1').then(function(cache) {
      return cache.addAll([
        '/cache-test/',
        '/cache-test/index.html',
        '/cache-test/style.css',
        '/cache-test/worker.js',
        '/cache-test/gallery/',
        '/cache-test/gallery/zdjecie.jpg',
        '/cache-test/gallery/logo.jpg'
      ]);
    })
  );
});
```

Jak widać mamy do czynienia z ustawieniem reakcji na zdarzenie install, wywoływane przy uruchomieniu całego mechanizmu. Wewnątrz metody obsługującej zdarzenie wywołujemy metodę już uruchomionego zdarzenia o nazwie waitUntil; pozwala ona na wydłużenie działania obsługi wywołanego zdarzenia zabezpieczając je przed zakończeniem go przez przeglądarkę zanim wykona swoją pracę. Parametrem waitUntil może być jedynie obiekt typu Promise - w naszym wypadku będzie to wartość zwracana z wywołania obiektu CacheStorage (dostępnego jako zmienna globalna caches). Tworzenie obiektu realizowane jest przez metodę open, która przyjmuje jako parametr dowolną nazwę reprezentującą dalej naszą pamięć podręczną.

UWAGA! Proszę pamiętać, że dla każdej ze stron możemy utworzyć kilka wersji pamięci podręcznej. Należy jednak mieć na uwadze by były one unikatowe w obrębie danej witryny.

Ponieważ nie przewidujemy porażki wykorzystujemy jedynie metodę then (sukces Promise). Wewnątrz then nakazujemy Promise wywołać utworzoną funkcję anonimową, która doda do naszej pamięci wskazane pliki i foldery.

INFORMACJA: Ważne jest by podać WSZYSTKIE FOLDERY, PODFOLDERY oraz PLIKI do zapisania podręcznego. Service Workers nie przewiduje tzw. dzikich kart (przynajmniej na chwilę obecną) pozwalających na dołożenie całej zawartości naszej witryny. Oczywiście problem ten można rozwiązać dodatkowymi bibliotekami, np. Service Worker Precache, jednak należy mieć na uwadze, że niekoniecznie muszą one działać ze wszystkimi przeglądarkami wspierającymi to rozwiązanie.

Kolejną częścią pliku będzie reakcja na dopasowanie zawartości pamięci do odpowiedzi ze strony:

```
this.addEventListener('fetch', function(event) {
  var response;
  event.respondWith(caches.match(event.request).catch(function() {
    return fetch(event.request);
  }).then(function(r) {
    response = r;
    caches.open('version1').then(function(cache) {
      cache.put(event.request, response);
    });
    return response.clone();
  }).catch(function() {
    return caches.match('/cache-test/gallery/logo.jpg');
  }));
});
```

Zdarzenie (event) jest tutaj silnie związane z otrzymaną przez przeglądarkę odpowiedzią od serwera WWW. Poprzez metodę respondWith (metoda interfejsu FetchEvent) zwracany jest bowiem obiekt Response (ze wszystkimi komunikatami odpowiedzi serwera). Jako parametr przyjmuje obiekt Promise będący, w powyższym przypadku, efektem wykonania metody match zmiennej caches (CacheStorage). Metoda ta ma za zadanie porównać nagłówek odpowiedzi ze zgromadzonymi odpowiedziami (czy strona przypadkiem nie znajduje się już w naszej pamięci podręcznej). Jeżeli jej nie ma wywołane zostanie zdarzenie fetch, które ma za zadanie obsłużyć nasze żądanie (np. wyświetlić stronę/błąd z danej przeglądarki internetowej bądź zgromadzić dane do trybu bezpołączeniowego). Jeżeli natomiast obiekt zakończy pracę z powodzeniem, wywołana przez niego funkcja uzupełni zadeklarowaną przez nas zmienną o wartość aktualnej odpowiedzi. Ostatnia obsługa błędu (catch) dodana jest na wypadek gdyby pamięć podręczna nie była kompletna – przykładowo wyświetlimy wtedy obrazek logo.jpg (o ile będzie dostępny; opieramy się tutaj o założenie, że jeżeli chociaż raz ktoś odwiedził naszą stronę to obraz ten musiał zostać zapisany w pamięci podręcznej).

W przypadku powodzenia (funkcja then) zwracana jest wartość response.clone(). Wbrew pozorom operacja ta jest dość istotna z punktu widzenia komunikacji serwer-klient. Jeżeli nie sklonowalibyśmy odpowiedzi serwera to przeglądarka, po uzupełnieniu pamięci podręcznej, nie wykonałaby już np. wyświetlenia zawartości strony (zrobiłaby to dopiero po ponownym odświeżeniu adresu – strumień odpowiedzi może być czytany tylko jeden raz). Dzięki metodzie clone() będzie to dla przeglądarki „nowa” odpowiedź, tyle co „otrzymana”.

Technologia pozwala na posiadanie większej ilości treści dla jednej strony. Jeżeli chcemy równolegle stworzyć kolejną wersję offline to musimy po prostu zmienić nazwę kontenera w parametrze metody open, np. na ‘version2’. Należy pamiętać, że nazwy te muszą być unikatowe w obrębie naszej strony WWW.

Ostatnią interesującą nas częścią Service Workers będzie usuwanie zbędnego schowka pamięci podręcznej. Do tego celu najbardziej adekwatna będzie obsługa zdarzenia activate. Service Workers wykorzystuje to zdarzenie w chwili, gdy np. tworzona jest nowa baza pamięci podręcznej, instalowanie jej bądź wyczekuje on na zakończenie któregoś z procesów Promise (waitUntil). Oczywiście usunięcie wskazanej wersji może odbyć się ręcznie (w innej części kodu) jednak stosownym jest zautomatyzować ten proces.

Usuwanie powala na lepszą gospodarkę przestrzenią pamięci przeglądarki, pewność wyświetlania najnowszej wersji strony oraz wymianę uszkodzonych plików na świeże, naprawione. Przykładowy kod usuwający niepotrzebną już wersję plików może wyglądać następująco:

```
this.addEventListener('activate', function(event) {
  var cacheWhitelist = ['version1', ‘version2’];
  event.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (cacheWhitelist.indexOf(key) === -1) {
          return caches.delete(key);
        }
      }));
    })
  );
});
```

Jak widać kod jest uniwersalny, a jego działanie niezwykle proste. Najpierw tworzymy listę, w której umieszczamy wersję (bądź kilka wersji) naszych wersji plików nie do kasowania. Następnie wywołujemy pętlę waitUntil, która jako parametr przyjmuje obiekt tworzony przez metodę keys() ze zmiennej caches (CacheStorage). W środku tego obiektu tworzony jest czysty obiekt Promise przepatrujący zwróconą listę pamięci podręcznej. Jeżeli dana pozycja zwrócona do keyList nie będzie widniała w utworzonej przez nas liście cacheWhitelist to jej zbiór zostanie w całości usunięty poprzez metodę caches.delete().

Należy mieć na uwadze, że przedstawione powyżej fragmenty Service Workers są wystarczające do utworzenia strony i/lub aplikacji działającej niezależnie od dostępu do sieci internet (bądź intranet). Nie jest to jednak kres możliwości tego rozwiązania – przedstawione obiekty posiadają dodatkowe właściwości i metody pozwalające na jeszcze większą kontrolę tego co zapisujemy, a co usuwamy z pamięci podręcznej.

W sieci można spotkać się ze stwierdzeniem, że podejście to jest znacznie przesadzone względem zastosowania. Faktycznie – dla prostej strony WWW rozwiązanie to wygląda ciężko zarówno w implementacji jak i użytkowaniu. Jednak w przypadku aplikacji biurowych, bazodanowych bądź gier rozwiązanie to jest bardzo dobre. Najlepszym rozwiązaniem jest napisać uniwersalny kod obsługi trybu offline i podłączyć go do swoich stron lub skorzystanie z gotowych rozwiązań dostępnych w sieci (np. UpUp do którego odnośnik znajduje się w materiałach).

2.2 Pamięć przeglądarki (HTML Local Storage)

Wraz z HTML5 swoją premierę miała nowa funkcjonalność przeglądarek WWW – lokalna przestrzeń zapisu danych. Już wcześniejsze wersje HTML borykały się z problemem zapisu danych dla użytkownika portalu/aplikacji WWW w mało wydajnych i nieefektywnych ciasteczkach, których obsługę użytkownicy często wyłączali (przez ich rzekome niebezpieczeństwo). Ciasteczka mają ten problem, że za ich pomocą można jedynie zapisać dane tekstowe nie przekraczające 4 KB (domyślna wielkość jednej linii tekstu w pliku tekstowym).

Rozwiązanie zaimplementowane w HTML5 przeznacza przestrzeń minimum 5 MB dla każdej strony WWW. Dane tego typu trzymane są wedle schematu klucz → wartość. Możliwa jest też opcja zapisu danych pod właściwościami obiektu localStorage/sessionStorage (jako zmienne). Prócz tradycyjnych danych tekstowych możliwy jest także zapis plików binarnych – trzeba jednak zachować mechanizm zapisu jaki stosuje poczta WWW (zamieniać dane poprzez metodę toDataURL() - dostępna dla wszystkich obiektów JavaScript).

UWAGA! Wielkość pamięci dla domeny w przeglądarce NIE JEST gwarantowana poprzez specyfikację. Oznacza to, że możemy dostać od 0 MB do nielimitowanego miejsce pod zapis danych. W materiałach znajduje się odnośnik do kodu sprawdzającego dostępną wielkości pamięci

dla naszej strony. Niektóre przeglądarki, jak np. Opera, pozwalają użytkownikom na zmianę dostępnej wielkości pamięci dla danej domeny – może to być dobre rozwiązanie w przypadku naprawdę rozbudowanych aplikacji HTML (szczególnie dla aplikacji intranet).

2.2.1 Pamięć sesyjna (sessionStorage)

Pamięć ta pozwala na zapamiętanie danych dla naszej witryny. Jej cechą charakterystyczną jest ulotność – dane w niej automatycznie niszczone są z chwilą zamknięcia karty strony (w najgorszym przypadku w chwili zamknięcia przeglądarki). Idealnie nadaje się do zapamiętania takich informacji jak nazwa zalogowanego użytkownika, danych chwilowo uzupełnianych w formularzu (w przypadku nieumyślnego odświeżenia strony) czy pisanej treści (np. aplikacje biurowe, fora internetowe).

2.2.2 Pamięć lokalna (localStorage)

Działanie tego obiektu jest podobne do poprzedniego. Różnica polega na tym, że dane w nim zapisane są automatycznie odkładane w bazie SQL przeglądarki. Dzięki temu nawet po jej wyłączeniu i ponownym włączeniu będziemy mogli korzystać z zapisanych informacji. Dane w tej części nie są zapisywane jeżeli użytkownik przegląda sieć poprzez tryb prywatny. W różnych przeglądarkach w różny sposób przyjęto rozwiązanie działania pamięci lokalnej dla wspomnianego trybu, dlatego najlepszych rozwiązaniem jest korzystać z pamięci sesyjnej, a pamięć lokalną wykorzystywać jeżeli jest dostępna (natomiast nie doprowadzać do sytuacji, w której opieramy się jedynie o pamięć lokalną).

2.2.3 Użycie pamięci podręcznej przeglądarki – przykładowy kod

```
localStorage.setItem("nazwisko", "Kowalski"); //zapisanie nazwiska w pamięci lokalnej
console.log(localStorage.getItem("nazwisko"); //odczyt nazwiska z pamięci lokalnej
localStorage.removeItem("nazwisko"); //usunięcie nazwiska z pamięci podręcznej
localStorage.imie = "Nina"; //zapisanie imienia przy użyciu zmiennej zamiast setItem()
```

Zapis do pamięci sesyjnej (sessionStorage) jest analogiczny – zamiast obiektu localStorage używamy sessionStorage.

3. Zadania do wykonania

Projektowaną stronę należy wyposażyć w mechanizm przechowywania w trybie offline. Celem przetestowania działania mechanizmu należy swój projekt umieścić na lokalnym serwerze WWW (np. XAMPP dla Windows/Linux/OS X).
Ponadto należy zaimplementować obsługę pamięci lokalnej w taki sposób, by nawet przypadkowe wyłączenie strony nie powodowało utraty informacji np. uzupełnionych już w formularzu kontaktowym. Zapisywane dane powinny być usuwane z chwilą naciśnięcia przycisku wyślij.

Proszę pamiętać, że powyżej znajdują się jedynie propozycje wykorzystania przedstawionych technologii. Implementacja ich ma jednak odpowiadać aktualnym potrzebom tworzonego projektu.

MATERIAŁY:
http://webroad.pl/html5-css3/732-manifest-html5-jak-stosowac-appcache
https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache
https://medium.com/@firt/service-workers-replacing-appcache-a-sledgehammer-to-crack-a-nut-5db6f473cc9b
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

https://developer.mozilla.org/en-US/docs/Web/API/Response
https://developer.mozilla.org/pl/docs/Web/API/Cache
http://stackoverflow.com/questions/34160509/options-for-testing-service-workers-via-http
https://developers.google.com/web/fundamentals/getting-started/primers/service-workers
https://www.talater.com/upup/
https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
http://stackoverflow.com/questions/2989284/what-is-the-max-size-of-localstorage-values
https://msdn.microsoft.com/pl-pl/library/hh580308(v=vs.85).aspx
http://stackoverflow.com/questions/19183180/how-to-save-an-image-to-localstorage-and-display-it-on-the-next-page
https://www.w3schools.com/html/html5_webstorage.asp