POLITECHNIKA ŁÓDZKA
Wydział Elektrotechniki, Elektroniki,
Informatyki i Automatyki

Praca dyplomowa inżynierska

Aplikacja webowa wspomagająca proces zapamiętywania

Emil Jasiński

Nr albumu: 164357

Opiekun pracy:
dr inż. Wojciech Tylman

Łódź, 2014

Spis treści

Streszczenie (Summary)

Aplikacja webowa wspomagająca proces zapamiętywania

Streszczenie po polsku.

TYTUŁ ANGIELSKI

Streszczenie po angielsku.

Wstęp

W obecnych czasach z każdej strony do naszego mózgu dociera masa zbędnych informacji. Część bez naszej woli jak na przykład świecący billboard reklamowy, na który mimowolnie spoglądamy jadąc samochodem, a na cześć decydujemy się sami gdy naszą pierwszą decyzją po przebudzeniu jest włączenie telewizora lub komputera.

Coraz więcej informacji jest nam również potrzebne w życiu zawodowym. Wpis o znajomości jednego języka obcego w CV nie wyróżnia nas już na tle innych ubiegających się o pracę. Niektóre profesje, jak na przykład zawód programisty wymagają nieustannego podnoszenia swoich kwalifikacji.

Taki nadmiar informacji, którego nie jesteśmy w stanie przetworzyć nazywa się przeciążeniem informacyjnym i może skutkować pogorszeniem się naszego samopoczucia i uczuciem przytłoczenia. Rozwiązaniem na ten rosnący problem społeczny jest wygospodarowanie każdego dnia czasu na spacer czy słuchanie muzyki, która nas odpręża. Ulgę naszemu organizmowi może też przynieść skrócenie czasu poświęconego na naukę. Okazuje się, że istnieje wiele sposobów na ułatwienie zdobywania nowej wiedzy. Są to między innymi:

— umiejętność szybkiego czytania
— mnemotechnika
— metoda SuperMemo

Ostatnia z wymienionych metod oraz aplikacja implementująca algorytm o tej samej nazwie są przedmiotem tej pracy.

Projekt aplikacji przedstawionej w rozdziale "Projekt aplikacji wspomagającej zapamiętywanie" powstał by w jak największym stopniu spełnić wymagania potencjalnych użytkowników. Założono, że użytkownik może chcieć korzystać z aplikacji na swoim smartphonie, tablecie czy komputerze, bez dostępu do Internetu gdy nie jest to niezbędne. Przy dodatkowym założeniu, że projekt nie wymaga dużej mocy obliczeniowej czy dostępu do natywnych elementów systemu operacyjnego, zdecydowano się na wykorzystanie zbioru standardów oferowanych przez HTML w wersji piątej.

Rozdział 1

Cel i zakres pracy

Celem pracy jest zaprojektowanie aplikacji, której pierwsza część pozwoli użytkownikowi utworzyć bazę wiedzy składającą się z pytań i odpowiedzi. Druga część służy do prezentowania pytania. Użytkownik ma na tym etapie zastanowić się nad odpowiedzią. Gdy zna już odpowiedź lub uznaje, że przypomnienie sobie odpowiedzi trwa zbyt długo, przechodzi do części prezentacji odpowiedzi z formularzem, w którym sam ocenia swój poziom zapamiętania tego pytania. Jest to sposób przyjęty w konkurencyjnych aplikacjach wykorzystujących algorytm SuperMemo.

Zakres pracy przedstawia się następująco:
— Test przeglądarek internetowych pod kątem obsługi HTML5
— Opis użytych technologii ze standardu HTML5
— Przegląd cech Visual Studio i ASP.NET, które ułatwiają pisanie nowoczesnych aplikacji webowych
— Historia algorytmu SuperMemo
— Implementacja algorytmu SuperMemo w wersji drugiej
— Zaprojektowanie aplikacji wykorzystującej algorytm
— Testy aplikacji

Rozdział 2

Mobilne aplikacje webowe

Rozdział opracowano na podstawie [1], [2], [3].

2.1. Historia specyfikacji HTML5

HTML (ang. HyperText Markup Language) jest językiem znaczników używanym do stworzenia struktury i wyglądu strony WWW (ang. World Wide Web). Pierwsza specyfikacja została upubliczniona w roku 1991 przez fizyka Tima Bernersa-Lee, który wcześniej stworzył prototyp hipertekstowego systemu informatycznego, pracując dla ośrodka CERN. Ideą, która zdecydowała o sukcesie takiego systemu prezentowania treści w sieci Internet były odnośniki. W ten sposób na jednej stronie można było zgrupować hiperłącza do wyników badań ośrodków naukowych z wielu miejsc na świecie.

Ostanią wersją specyfikacji miała być specyfikacja HTML 4.01, która została udostępniona przez organizację W3C (ang. World Wide Web Consortium) w roku 1999, jednak już w 1998 W3C podjęło decyzje o zatrzymaniu jej rozwoju i rozpoczęciu prac nad bazującą na XML alternatywą nazwaną XHTML. Kontrowersje wzbudziła specyfikacja XHTML 2.0, która nie była kompatybilna z wcześniejszymi wersjami.

W roku 2004, pracownicy firm Apple, Mozilla Foundation i Opera założyli grupę WHATWG (ang. Web Hypertext Application Technology Working Group). Było to spowodowane obawami przed kierunkiem rozwoju jaki przyjęło W3C - brak rozwoju HTML i brak odpowiedzi na potrzeby twórców witryn i przeglądarek internetowych. Grupa zaczęła pracować nad standardem HTML w wersji piątej.

W roku 2006 W3C zgodziło się użyć propozycji grupy WHATWG. Grupa W3C powołana do rozwoju XHTML 2 zaprzestała prac w 2009. W3C postanowiło przyspieszyć rozwój HTML 5 inwestując w pracę nad nim swoje zasoby.

W roku 2014 HTML 5 stanie się rekomendacją W3C. Na rok 2016 planowane jest opublikowanie HTML 5.1.

2.2. Różnice w stosunku do HTML4

HTML 5 jest stale rozwijającym się standardem. Projektując stronę używając HTML 4 nie musimy martwić się o aktualizowanie strony, o ile oczywiście nie chcemy dodać nowych treści lub funkcjonalności. Strona zbudowana w oparciu o niego, która przeszła proces walidacji powinna wyświetlać się identycznie na wszystkich przeglądarkach niezależnie od daty powstania. Jeśli przejdziemy na HTML 5 może dojść do sytuacji, że wykorzystany przez nas kilka miesięcy wcześniej element lub atrybut zmienił się. Dotyczy to szczególnie elementów wprowadzonych w HTML 5 (np. znaczniki audio, video). W związku z tym uproszczeniu mogła ulec deklaracja typu dokumentu.
Zamiast

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

używamy:

```
<!DOCTYPE html>
```

Nowe znaczniki to między innymi:

canvas używany do rysowania grafiki przez skrypty pisane w języku JavaScript
audio definiuje plik dźwiękowy jako treść strony
video definiuje plik wideo jako treść strony
article definiuje artykuł
nav definiuje sekcję, która zawiera tylko linki
address definiuje sekcję zawierającą informacje kontaktowe
main definiuje główną, najbardziej znaczącą część strony
dialog definiuje okno dialogowe
section definiuje sekcje dokumentu
footer definiuje stopkę dla całego dokumentu lub sekcji
header definiuje nagłówek dla całego dokumentu lub sekcji
mark definiuje zaznaczony tekst
progress definiuje postęp zadania
time definiuje datę/czas

API (ang. Application Programming Interface) HTML 5 udostępnia programiście wiele nowości (rys. 2.1).

Ciasteczka (ang. cookies) to porcja danych wysyłana z przeglądarki internetowej i przechowywana jako mały plik tekstowy na dysku twardym użytkownika strony. Wykorzystywane są do przechowywania informacji o aktywności użytkownika (np. czy

Rysunek 2.1: Obsługa poszczególnych technologii HTML5 na najpopularniejszych przeglądarkach
Źródło: fmbip.com/litmus/

użytkownik jest zalogowany).

Zalety:

— nie wymagają zasobów serwera,

— są łatwe w implementacji.

Wady:

— są każdorazowo wysyłane z żądaniem i odpowiedzią serwera,

— mogą być wyłączone przez użytkownika przeglądarki,

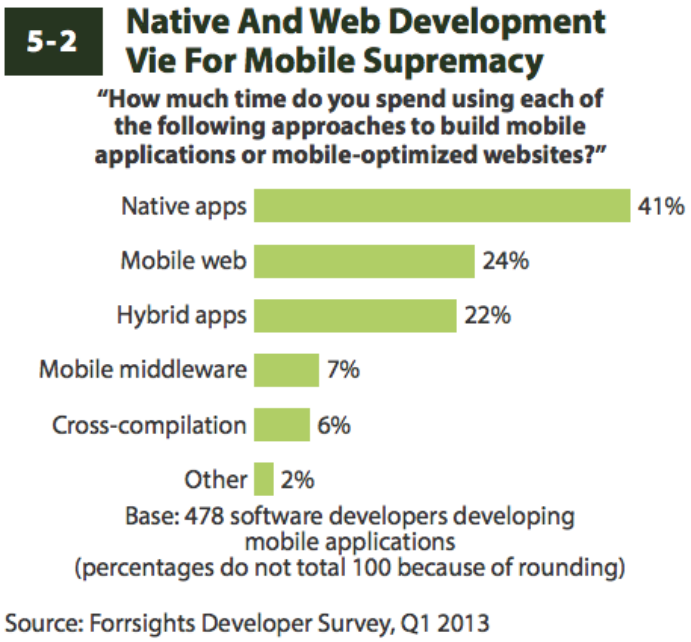
— problemy związane z bezpieczeństwem,

— mała pojemność.

Nowa specyfikacja proponuje alternatywę w postaci WebStorage oraz IndexedDB. WebStorage dzieli się na local storage i session storage. Dane zapisane w local storage nie wygasają. Dane w session storage wygasają po zamknięciu przeglądarki lub okna oraz są oddzielne dla każdej z instancji aplikacji.

IndexedDB jest bardziej zaawansowanym rozwiązaniem. Jest to interfejs lokalnej bazy danych, w której przechowywane mogą być zarówno wartości proste jak i skomplikowane, hierarchiczne obiekty. Rozwiązanie zostało zaproponowane przez firmę Oracle w roku 2009.

Zupełną nowością jest natomiast możliwość pisania aplikacji działających w trybie offline. AppCache pozwala twórcy witryn zdefiniować, które pliki powinny zostać zapisane na dysku użytkownika, a następnie być dostępne w trybie offline. Aplikacja napisana w ten



Rysunek 2.2: Popularność sposóbów budowy aplikacji mobilnych
Źródło: www.forrester.com

sposób załaduje się i będzie działać poprawnie nawet gdy użytkownik jest poza zasięgiem sieci pod warunkiem, że już kiedyś odwiedził daną witrynę. Można więc przyrównać pierwsze władowanie strony w przeglądarce do instalacji natywnych aplikacji (np. w sklepie Google Play w systemie Android). By następnie zaktualizować zapisane na dysku pliki należy zmienić manifest (plik tekstowy, który zawiera między innymi spis plików).
Zalety:
— użytkownicy mogą nawigować po stronie bez dostępu do internetu
— zysk na szybkości - zasoby naszej aplikacji ładowane są bezpośrednio z dysku

Powyższe cechy powodują, że warto przed rozpoczęciem pisania aplikacji na urządzenia mobilne przemyśleć zalety i wady wybrania podejścia - zaprojektuj raz, uruchamiaj wszędzie, oferowanego przez HTML5 i język JavaScript. Zainteresowanie programistów tą technologią możemy odczytać z rys. 2.2.

2.3. RWD (Responsywny wygląd)

RWD (ang. Responsive web design) - responsywny wygląd - jest to podejście w projektowaniu witryn internetowych, którego celem jest uzyskanie jak najlepszych doświadczeń dla odwiedzającego stronę. Strony napisane w oparciu o tę technikę są łatwe do nawigacji, czytania oraz obsługują większość urządzeń z dostępem do internetu(od urządzeń mobilnych do komputerów PC).

Strona zaprojektowana w RWD przystosowuje szablon do środowiska, w którym jest oglądana poprzez użycie płynnych, bazujących na proporcjach kontenerach, grafikach o

zmiennych rozmiarach oraz funkcjonalności kaskadowych arkuszy stylów(CSS3) - media queries .

Gdy projektujemy nową aplikację webową warto przyjąć podejście o nazwie "mobile first". Zakłada ono, że pierwszym szablonem, który zostanie zaprojektowany jest ten na urządzenia mobilne. Na tym widoku powinny znaleźć się najbardziej potrzebne elementy. Taki szablon powinien być czytelny i wyraźny, by można go było obsługiwać bez problemów palcami na ekranie urządzenia mobilnego o małej rozdzielczości ekranu. Następnie powinniśmy się zastanowić jakie elementy chcielibyśmy pokazywać dodatkowowo w miarę zwiększania rozdzielczości ekranu. Na większych rozdzielczościach możemy też założyć, że użytkownik będzie używał myszy komputerowej, a więc będzie mógł obsługiwać nawet małe elementy na stronie.

By wykorzystać zalety tego podejścia należy dodać dodatkowy znacznik w sekcji HEAD naszej aplikacji:

```
<meta name='viewport' content='width=device-width, initial-scale=1.0'>
```

Jest to wskazówka dla przeglądarki internetowej, że nie powinna ona skalować danej strony.

W zależności od rozdzielczności ekranu projektant witryny internetowej może zdecydować o wczytywaniu innych styli. Gdy chcemy wczytać style dla rozdzielczności powyżej 900 pikseli szerokości, ale mniejszej niż 1024 powinniśmy użyć:

```
@media screen and (min-width: 900px) and (max-width: 1024px)
{
        /* style tylko dla tej rozdzielczosci */
}
```

2.4. Visual Studio i ASP.NET jako środowisko pracy

Wybierając edytor lub zintegrowane środowisko programistyczne do budowy aplikacji webowej warto sprawdzić czy posiada on zestaw pluginów Zen Coding. Znacząco przyspiesza to pisanie kodu HTML, a dzięki temu, że silnik pluginów jest niezależny od edytora powinniśmy znaleźć dodatek do każdego popularnego IDE (ang. Integrated Development Environment). Użycie tego dodatku wiąże się z wpisaniem w edytor odpowiedniej komendy i naciśnięciu klawisza TAB.

Przykład działania:

```
html:5>div#page>div.logo+nav>ul#navigation>li*5>a
```

Rezultat:

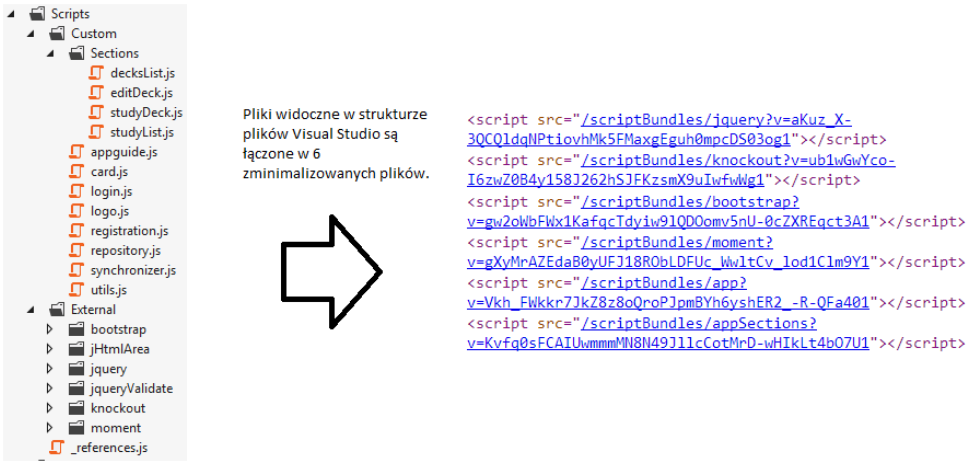
```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
4     <title></title>
5     <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
6 </head>
7 <body>
8     <div id="page">
9         <div class="logo"></div>
10        <nav>
11            <ul id="navigation">
12                <li><a href=""></a></li>
13                <li><a href=""></a></li>
14                <li><a href=""></a></li>
15                <li><a href=""></a></li>
16                <li><a href=""></a></li>
17            </ul>
18        </nav>
19    </div>
20 </body>
21 </html>
```

Odpowiedni dodatek do Visual Studio można pobrać przez wbudowane narzędzie do pobierania dodatków jak i bibliotek programistycznych - NuGet. Natomiast funkcjonalnością, której nie możemy odtworzyć w innych IDE jest IntelliSense. Dzięki niej podczas wpisywania kodu w języku C# czy JavaScript widzimy podpowiedzi odnośnie nazwy funkcji czy opis parametrów jak dana metoda przyjmuje. W przypadku tego drugiego języka podpowiedzi w innych IDE są rzadkością.

W nowoczesnych aplikacjach na znaczeniu zyskała warstwa kliencka, a cały wygląd powinien być zdefiniowany poprzez kaskadowe arkusze stylów. Zasadą, która pozwala na oddzielenie kodu html, css i js jest tworzenie dla nich osobnych plików. Powinny być ułożone w logiczny sposób np. każda klasa napisana w języku JavaScript w osobnym pliku. Wiąże się z tym zwiększenie liczby żądań do serwera. Istnieje cecha ASP.NET dzięki której możemy zachować logiczną strukturę naszych plików warstwy klienckiej i jednocześnie zwiększyć wydajność naszej witryny. Bundles, bo tak nazywa się ta funkcjonalność, pozwala nam utworzyć grupę plików, która zostanie połączona w jeden plik o podanej przez nas nazwie. Dodatkowo plik wynikowy zostanie zminimalizowany, co może znacząco obniżyć jego rozmiar. Minimalizację plików może wyłączyć w trybie DEBUG.

Przeglądarki internetowe posiadają pamięć podręczną tzw. cache, który przechowuje pliki z wcześniej odwiedzonej strony. W ten sposób łącze internetowe klienta i zasoby serwerowe są odciążone, a strona ładuje się szybciej. Jest jednak istotna wada tego rozwiązania - klient może posiadać nieaktualny plik. Bundles oferują rozwiązanie dzięki dołączaniu do adresu z plikiem argumentu tzw. Content Hash. Jest on generowany na podstawie treści pliku. Gdy treść ulegnie zmianie zmieni się też hash. W ten sposób klient pobierze zawsze najnowszy plik. Jest to ważne gdy w naszych plikach znajduje się rozwiązanie krytycznego błędu naszej aplikacji.



Rysunek 2.3: Widok plików w IDE oraz kod html połączonych plików
Źródło: Opracowanie własne

Powyższe cechy sprawiają, że Visual Studio i ASP.NET jest dobrym wyborem jeśli szukamy środowiska do pisania nowoczesnych aplikacji webowych.

Rozdział 3

Algorytm SuperMemo

Rozdział opracowano na podstawie [4], [5], [6], [7], [8].

3.1. Krzywa zapominania

Inaczej zwana krzywą Ebbinghausa, od nazwiska Hermanna Ebbinghausa. Ten niemiecki psycholog zaproponował w swojej pracy "O pamięci" zależność ilości pamiętanego materiału od czasu. Typowy wykres krzywej zapominania pokazuje, że ludzie zapominają połowę nowo nabytej wiedzy w ciągu kilku dni lub tygodni jeśli materiał nie był regularnie powtarzany. W 1885 Ebbinghaus wysnuł teorię o wykładniczej naturze zapominania i opisał ją wzorem:

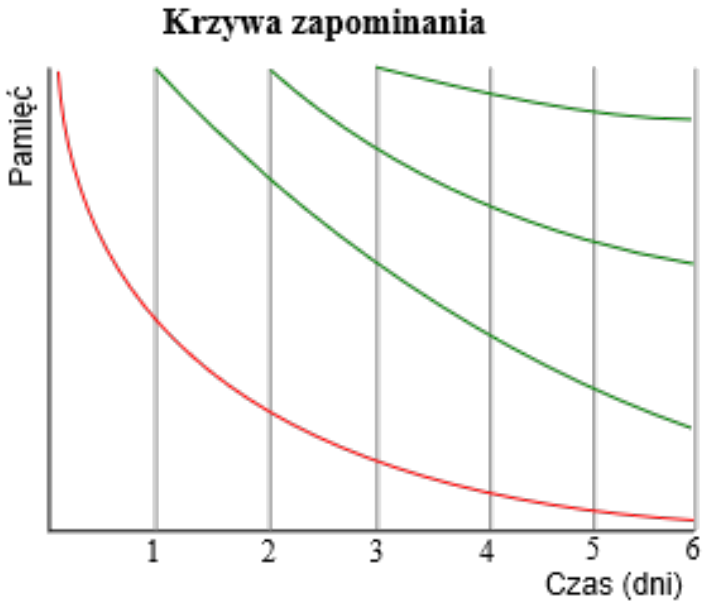
$$R = e^{\frac{-t}{s}},$$

gdzie:
R - zachowanie wiedzy
S - względna siła pamięci
t - czas

Ebbinghaus uczył się sylab pozbawionych znaczenia poprzez powtarzanie po różnych okresach czasu. Ustalił w ten sposób, że spadek zapamiętanych informacji staje się coraz mniejszy(rys. 3.1). Krzywa zapominania stała się fundamentem do dalszych badań i w rezultacie opracowanie algorytmu SuperMemo.

3.2. Opis

Dr Piotr Woźniak prowadzi badania nas teorią metod nauczania, które zaowocowały powstaniem algorytmu SuperMemo i programu o tej samej nazwie. Jest on też osobą odpowiedzialną za komercyjne wykorzystanie tej metody. Pierwsza wersja programu powstała w 1987 roku. Napisana została w środowisku Turbo Pascal 3.0, IBM PC i powstała by zaprezentować metodę na dwa podstawowe sposoby:
— zastosowanie procedur optymalizacji na najmniejszych możliwych przedmiotach
— zróżnicowanie elementów na postawie poziomu trudności



Rysunek 3.1: Wykres krzywych zapominania dla kolejnych powtórzeń
Źródło: Wikipedia

Zaproponowano w niej następujący wzór na obliczenie kolejnych przerw między powtórzeniami:

$$\begin{aligned} I(1) &= 1 \\ I(2) &= 6 \\ I(n) &= I(n-1) * EF \quad \text{, dla } n > 2 \end{aligned}$$

gdzie:
I(n) - przerwa bez powtórzenia po n-tym powtórzeniu (wyrażona w dniach)
EF - ang. easiness factor - odzwierciedla poziom trudności w zapamiętaniu i przypomnieniu danego faktu

Wartość EF może wynosić od 1,1 dla najtrudniejszego faktu do 3,5 dla najprostszego. W momencie wprowadzenia faktu do bazy SuperMemo wartość EF jest równa 2,5. Jeżeli w procesie powtarzania materiału istnieją problemu z przypomnienim sobie odpowiedzi na dane pytanie wartość ta maleje.

Wkrótce po utworzeniu pierwszej wersji SuperMemo, autor zauważył, że wartość EF nie powinna spadać ponizej 1,3, gdyż powodowało to, że pytania wyświetlały się zbyt często.

By obliczyć nową wartość EF, uczeń musi po każdej odpowiedzi na pytania ocenić swój poziom przyswojenia danego faktu. Program SuperMemo używa skali od 0 do 5. Ogólna forma użytego wzoru wygląda następująco:

$$EF' = f(EF, q)$$

gdzie:
EF' - nowa wartość EF

EF - stara wartość EF
q - jakość odpowiedzi
f - funkcja użyta do obliczenia EF'

Funkcja obliczania EF':

$$EF' = EF + (0,1 - (5 - q) * (0,08 + (5 - q) * 0,02))$$

Warto zauważyć, że dla q równego 4 wartość EF nie ulega zmianie.

3.3. Algorytm SuperMemo w wersji drugiej

Algorytm SuperMemo w wersji drugiej programu przedstawia się następująco:

1. Podziel wiedzę, którą chcesz przyswoić w jak najmniejsze fakty(pytanie i odpowiedź).
2. Przypisz dla wszystkich faktów wartość EF równą 2,5.
3. Powtarzaj fakty robiąc następujące przerwy:

$$\begin{aligned} I(1) &= 1 \\ I(2) &= 6 \\ I(n) &= I(n-1) * EF \quad \text{, dla } n > 2 \end{aligned}$$

4. Po każdym powtórzeniu oceń jakość odpowiedzi w skali od 0 do 5, gdzie
   5 - doskonała odpowiedź
   4 - poprawna odpowiedź po zastanowieniu
   3 - poprawna odpowiedź po długim zastanowieniu
   2 - niepoprawna odpowiedź, gdy poprawna wydaje się łatwa do przypomnienia
   1 - niepoprawna odpowiedź, poprawna została przypomniana
   0 - brak odpowiedzi

5. Po każdej odpowiedzi zmodyfikuj wartość EF przypisaną do danego faktu według wzoru:

$$EF' = EF + (0,1 - (5 - q) * (0,08 + (5 - q) * 0,02))$$

   Jeśli ocena odpowiedzi jest mniejsza niż 1,3 to ustaw nową wartość EF na 1,3.
6. Jeśli ocena odpowiedzi jest niższa niż 3 to zacznij powtórkę dla faktu bez zmiany wartości EF.
7. Po każdym powtórzeniu w danym dniu powtarzaj elementy ocenione poniżej 4. Kontynuuj dopóki wszystkie fakty będą ocenione przynajmniej na 4.

Algorytm użyty w aplikacji, która jest przedmiotem tej pracy, zawiera się w klasie Card i wygląda następująco:

```
var Card = function (deckId, front, back) {
    var self = this;
    self.data = {
        deckId: deckId,
        front: front,
        back: back,

        repetitionCount: 0,
        easinessFactor: 2.5,
        nextRepetitionDate: Date.now(),
        daysToNextRepetition: 0
    };
};

Card.makeReview = function (data, grade) {
    if (grade < 3) {
        data.repetitionCount = 0;
        data.nextRepetitionDate = Date.now();
        data.daysToNextRepetition = 0;
    }
    else {
        ++data.repetitionCount;

        if (data.repetitionCount == 1)
            data.daysToNextRepetition = 1;
        else if (data.repetitionCount == 2)
            data.daysToNextRepetition = 6;
        else
            data.daysToNextRepetition
                = parseInt(data.daysToNextRepetition * data.easinessFactor);

        daysToNextRepetition = data.daysToNextRepetition;
        data.nextRepetitionDate = moment(data.nextRepetitionDate)
                                .add('days', data.daysToNextRepetition).valueOf();
    }

    data.easinessFactor = data.easinessFactor +
                                (0.1 - (5 - grade) * (0.08 + (5 - grade) * 0.02));

    if (data.easinessFactor < 1.3)
        data.easinessFactor = 1.3;
};
```

3.4. Programy wykorzystujące metodę SuperMemo

Najpopularniejsze programy wykorzystujące opisany algorytm to program SuperMemo UX oraz Anki.

Autorem SuperMemo UX jest dr Piotr Woźniak, który jest również autorem algorytmu i metody SuperMemo. Wersja dostępna dla użytkowników pojawiła się w 1991 roku. Model biznesowy programu opiera się na płatnych kursach językowych. Długi okres rozwoju aplikacji zaowocował w zaawansowane funkcje, takie jak:

— aktywacja i dezaktywacja informacji do nauki,
— moduł pozwalający na doskonalenie wymowy słów,
— obsługa jeden instancji programu przez wielu użytkowników,
— funkcja "incremental video" - nauka na podstawie filmów z serwisu YouTube.

Anki reprezentuje odmienne podejście. Program ten jest w pełni darmowy i oferuje bazy danych utworzone przez użytkowników. Autor pracuje nad rozwojem wersji na PC, webowej oraz przenaczonej na urządzenia z systemem operacyjnym iOS (iPhone / iPad / iPod Touch). Kod źródłowy aplikacji jest dostępny na otwartej licencji. Dzięki temu powstała niezależna wersja na urządzenia z systemem Android.

Rozdział 4

Projekt aplikacji wspomagającej zapamiętywanie

4.1. Założenia

Aplikacja ma stanowić alternatywę dla innych programów wspomagających proces nauki poprzez inteligentne planowanie powtórek. Zaletami w stosunku do programu autora algorytmu SuperMemo jest prostota obsługi i darmowe korzystanie. Cechami pozwalającymi konkurować z programem Anki są intuicyjna obsługa oraz fakt, że można ją odpalić na każdej platformie mobilnej jak i komputerze PC.

Multiplatformowość jest zapewniona poprzez napisanie aplikacji przeglądarkowej. W ten sposób aplikacja może być odpalona na platformach mobilnych, takich jak Windows Phone, Android, iPhone, mało popularnych platform takich jak Sailfish, Tizen, FirefoxOS, ale też na systemach, które powstaną w przyszłości o ile będą posiadały nowoczesną przeglądarkę internetową, co jest standardowym wyposażeniem każdej platformy.

Interfejs graficzny napisany jest tak, by dopasować się do rozdzielczości każdego urządzenia dostępnego na rynku.

4.2. Wymagania

Do korzystania z aplikacji wymagana jest przeglądarka internetowa obsługująca następujące funkcjonalności HTML 5:

— Local Storage

— Canvas

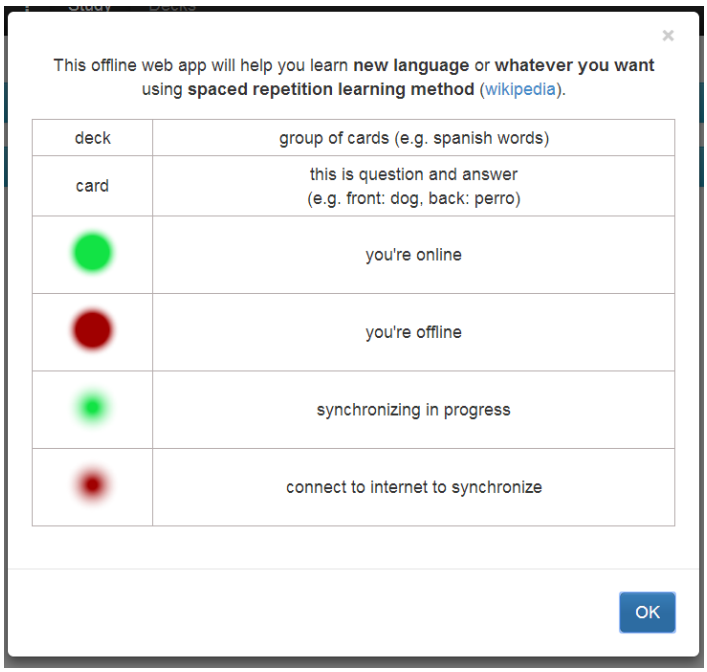
— IndexedDB

Są to najpopularniejsze przeglądarki na komputery PC: Google Chrome, Internet Explorer, Mozilla Firefox i Opera w wersjach najświeższych oraz ich odpowiedniki na urządzenia mobilne.

Aby synchronizować dane z serwerem konieczne jest posiadanie konta.

4.3. Instrukcja dla użytkownika

Przy pierwszym uruchomieniu aplikacji użytkownikowi wyświetla się okno z instrukcją dotyczącą aplikacji(rys. 4.1). Informuje ono użytkownika, że ma do czynienia z aplikacją



Rysunek 4.1: Instrukcja obsługi aplikacji
Źródło: Opracowanie własne

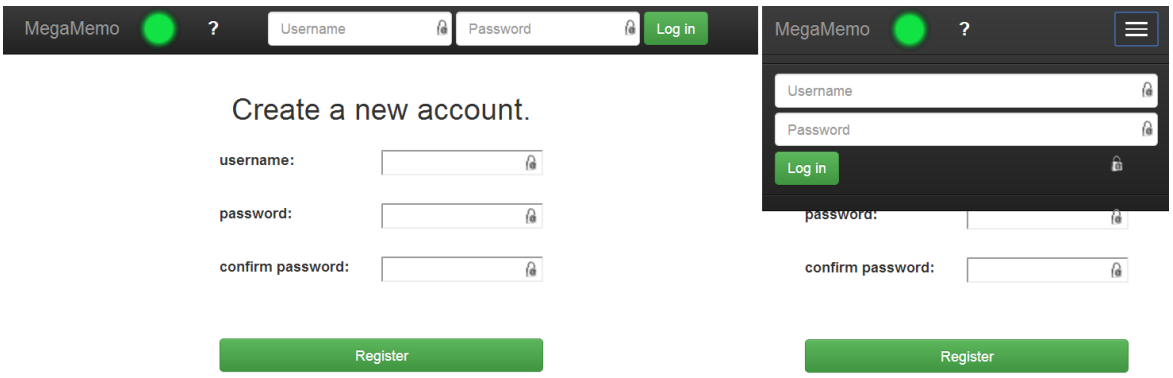
webową działającą w trybie offline, która ma za zadanie pomóc w nauce nowego języka, ale również w przyswojeniu innej wiedzy używając metody inteligentnego planowania następnych powtórek materiału. Więcej informacji o tej metodzie użytkownik może zdobyć klikając na link do artykułu na wikipedii. Pomoc zawiera także tabelę z wyjaśnieniem pojęć i symboli używanych w aplikacji. Dowiaduje się z niej, że deck(ang. talia) powinien traktować jako grupę pytań i odpowiedzi, którą pragnie przyswoić. Mogą być to na przykład hiszpańskie słowa z ich tłumaczeniem w języku angielskim. Card(ang. karta) stanowi pojedyńcze pytanie i odpowiedź(np. przód karty - dog, tył karty - perro). Przedstawiony jest również symbol, który reprezentuje statusy w jakich może pracować aplikacja:

— zielony - jesteśmy podłączeni do internetu, dane są zsynchronizowane z serwerem
— czerwony - nie mamy połączenia z internetem, dane są zsynchronizowane z serwerem
— zielony, pulsujący - trwa synchronizacją danych z serwerem
— czerwony pulsujący - po podłączeniu do intenetu rozpocznie się synchronizacja danych

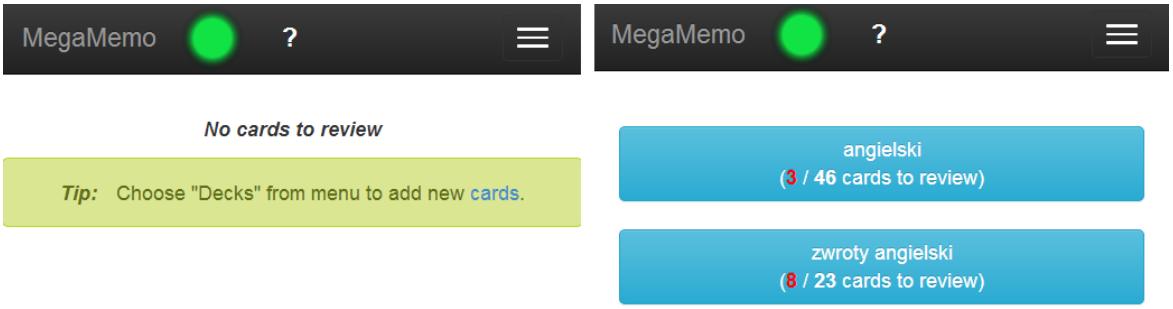
Identyczne okno pojawia się po kliknięniu na wyrazy ”deck” i ”card” w reszcie aplikacji.

Po zamknięciu okna użytkownik widzi formularz do założenia nowego konta na stronie lub w przypadku gdy już posiada konto może się zalogować poprzez formularz dostępny, w przypadku urządzenia mobilnego, po kliknięciu w przycisk umieszczony w prawym górnym rogu ekranu. W przypadku urządzeń z większą rozdzielczością ekranu pola formularza potrzebne do zalogowania są widoczne w tym miejscu(rys. 4.2).

Gdy wypełnimy pola formularza rejestracji i nie otrzymamy komunikatu o błędzie informującym o zajętej nazwie użytkownika lub niezgodności wprowadzonych haseł, zostaniemy automatycznie zalogowani na nowo utworzone konto. Domyślnym ekranem



Rysunek 4.2: Zrzut ekranu z formularzem zakładania nowego konta. W wersji komputerowej po lewej stronie i mobilnej po prawej.
Źródło: Opracowanie własne



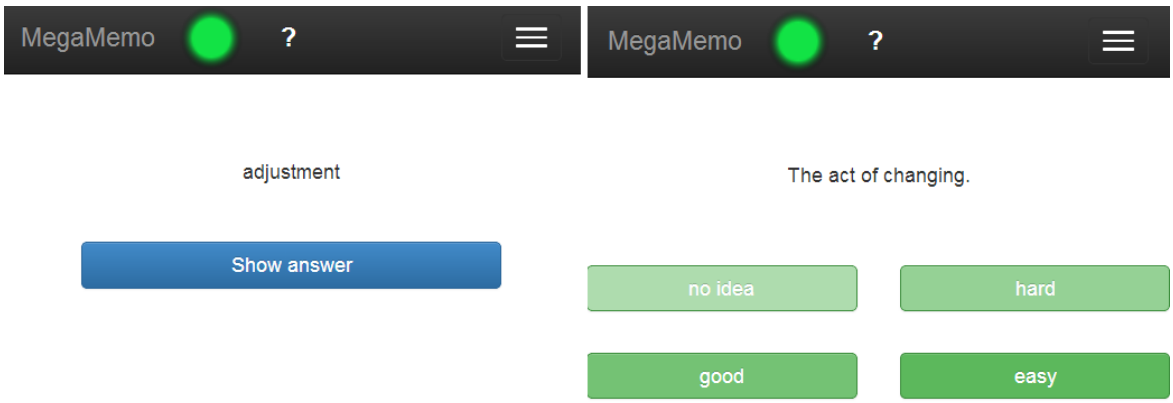
Rysunek 4.3: Zrzut ekranu z widokiem listy grup faktów do nauki. Wersja mobilna.
Źródło: Opracowanie własne

startowym jest lista grup faktów do nauki na obecny dzień(rys. 4.3). W przypadku gdy nie ma nic do powtarzania na dany dzień użytkownikowi wyświetla się komunikat z odpowiednią informacją oraz wskazówka, by kliknął odnośnik ”Decks” aby dodać nowe grupy do nauki i karty. Po kliknięciu na nazwę danej grupy przechodzimy do trybu powtarzania zawartych w niej faktów. Gdy algorytm dojdzie do miejsca, w którym nie ma już faktów do nauki przechodzimy z powrotem do listy.

Jeśli użytkownik chce dodać nowe fakty do swojej bazy wiedzy powinien przejść do pozycji ”Decks” w menu. Tam wyświetli się lista z już istniejącymi grupami lub informacja jak dodać nowe. Po kliknięciu w nazwę grupy dodajemy do niej nowe karty, aby to zrobić należy uzupełnić pola front - przednia część karty i back - tylnia część. Front to pytanie, back - odpowiedź na nie. Jeśli chcemy jednocześnie dodać dwie karty, jedną z pytaniem w polu front oraz odpowiedzią w back i drugą z pytaniem w polu back oraz odpowiedzią w polu front, należy zaznaczyć opcję o nazwie ”reverse”.

Ostatnim ekranem, na którym użytkownik spędzi najwięcej czasu i który jest główną funkcjonalnością aplikacji jest formularz powtarzania wiedzy. Schemat powtarzania wygląda następująco:

— Wyświetlone zostaje pytanie.
— Użytkownik zastanawia się nad odpowiedzią.



Rysunek 4.4: Zrzut ekranu formularza do powtarzania bazy wiedzy. Wersja mobilna.
Źródło: Opracowanie własne

— Gdy przypomina sobie odpowiedź lub dochodzi do wniosku, że nie jest w stanie sobie przypomnieć wybiera przycisk z napisem "Show answer" - "Pokaż odpowiedź". Następnie ma za zadanie ocenić jakość udzielonej odpowiedzi lub jej brak. Robi to przez kliknięcie odpowiedniego przycisku(rys. 4.4).

4.4. Zapis i synchronizacja danych

Wszelkie operacje dodania nowych danych są realizowane za pomocą metod zawartych w klasie "Repository". Służy ona do komunikacji z bazą danych po stronie klienta-przeglądarki - IndexedDB oraz klasy "Synchronizer".

```
var openRequest = indexedDB.open("mmdb", 2);

openRequest.onupgradeneeded = function (e) {
    console.log("Upgrading...");

    var thisDB = e.target.result;

    if (!thisDB.objectStoreNames.contains("decks")) {
        thisDB.createObjectStore("decks", {keyPath:"id",autoIncrement:false});
    }

    if (!thisDB.objectStoreNames.contains("cards")) {
        thisDB.createObjectStore("cards", {keyPath:"id",autoIncrement:false });
    }
}

openRequest.onsuccess = function (e) {
    db = e.target.result;

    db.onerror = function (event) {
        alert("Database error: " + event.currentTarget.target.errorCode);
    }
```

```
23
24      appStart();
25  }
```

Przed użyciem bazy IndexedDB należy najpierw otworzyć bazę. W lini 1 wywołujemy asynchronicznie metodę open z argumentem "mmdb" - nazwa bazy danych oraz 2 - wersja. Argumentem jest obiekt typu IDBRequest. W przypadku gdy na dysku użytkownika nie ma jeszcze bazy lub jest w niżej wersji wywołana zostanie metoda "onupgradeneeded". Po jej wykonaniu utworzone zostaną dwie tabele w bazie o nazwie "cards" oraz "decks". W obu przypadkach definiujemy indeks id, w którym wyłączamy automatyczne podnoszenie o jeden wartości, gdyż wartości id nadajemy na poziome bazy danych po stronie serwera. W przypadku gdy powiedzie się zaktualizowanie lub utworzenie bazy oraz otworzenie bazy wywołana zostaje funkcja przypisana do pole onsuccess obiektu IDBRequest. Ustawiamy w niej obiekt potrzebny do dalszych operacji na bazie IndexedDB podczas trwania korzystania z aplikacji. Ustawiamy również domyślną akcję, która wykona się przy wystąpieniu błędu zapisu lub pobrania danych. W takim przypadku wyświetlamy kod błędu na konsoli programisty JavaScript.

```
1  self.getDecks = function (callback) {
2      var trans = db.transaction("decks");
3      var store = trans.objectStore("decks");
4      var items = [];
5
6      trans.oncomplete = function (evt) {
7          callback(items);
8      };
9
10     var cursorRequest = store.openCursor();
11
12     cursorRequest.onsuccess = function (evt) {
13         var cursor = evt.target.result;
14         if (cursor) {
15             items.push(cursor.value);
16             cursor.continue();
17         }
18     };
19 }
```

"getDecks" jest jedną z metod służącą jedynie do pobrania danych. Jako argument przyjmuje funkcję, która wykona się po poprawnym pobraniu danych i przyjmuje jako argument pobrane dane. W ten sposób możemy wykonać kolejno po sobie akcje: akcja użytkownika, pobranie danych, przetworzenie danych(np. wyświetlenie na liście) mimo asynchronicznego wykonania kodu. Aby wykonać samą operację pobierania najpierw potrzebny nam obiekt IDBTransaction z parametrem nazwy transakcji. Domyślnie transakcja ma prawa czytania, które w tym przypadku są wystarczające, ponieważ nie modyfikujemy

danych. Następnie pobieramy obiekt dzięki któremu mamy dostęp do kursora znanego z tradycyjnych systemów zarządzania relacyjnymi bazami danych jak np. Microsoft SQL Server. Otwierając kursor implikujemy kolejne wykonania metody "onsuccess". Możemy z niego odczytać obiekt, wcześniej zapisany do naszej bazy. Dodajemy go do tablicy "items". Gdy wszystkie obiekty zostaną w niej zapisane, zostaje wywołana funkcja "oncomplete". W niej powracamy do fragmentu kodu, do którego działania niezbędne są pobrane dane.

```
1  self.addDeck = function (deck) {
2      var trans = db.transaction("decks", "readwrite");
3      var store = trans.objectStore("decks");
4
5      deck.id = self.getNextDeckId();
6      deck.lastUpdateDate = Date.now();
7
8      self.setDeckLastUpdateDate(deck.lastUpdateDate);
9      var request = store.add(deck);
10
11     request.onsuccess = function (e) {
12         synchronizer.decksToSync.push(deck);
13     };
14 };
```

Drugim typem metod występujący w obiekcie "repository" są funkcje modyfikujące lub dodające nowe obiekty do bazy. W "addDeck", tak jak w przypadku "getDecks" potrzebujemy obiektu transakcji. Dodatkowo musimy zaznaczyć, że chcemy modyfikować dane poprzez argument "readwrite". W argumencie funkcji przekazujemy obiekt, który chcemy zapisać. Po nadaniu id oraz daty ostatniej modyfikacji rozpoczynamy zapis. Jeśli się powiedzie wykona się metoda "onsuccess". W niej wywołujemy metodę obiektu "synchronizer", który jest odpowiedzialny za synchronizację danych z serwerem.

Koniecznym elementem niezbędnym dla aplikacji webowej działającej w trybie offline jest mechanizm zapewniającym synchronizacją danych z bazą danych po stronie serwera. Służy do tego klasa "Synchronizer". Jej metoda "setNetworkStatus" jest wywoływana co 1500 milisekund i sprawdza status połączenia z internetem poprzez odwołanie się do właściwości onLine obiektu navigator. W przypadku gdy jest ona ustawiona na wartość "false" możemy z pewnością założyć brak dostępu do internetu. W przypadku wartości "true" nie mamy tej gwaracji od wszystkich twórców przeglądarek internetowych. W związku z tym należy upewnić się poprzez testowe odwołanie się do dowolnego serwera. W przypadku powodzenia komunikujemy użytkownikowi, iż znajduje się w trybie online. W przypadku błędu lub przekroczenia maksymalnego czasu połączenia sygnalizujemy tryb offline.

Ustawienie wartości odpowiadającej stanowi połączenia internetowego klienta nie służy jedynie do celów infromacyjnych. Wartość ta jest niezbędna przy sprawdzeniu czy można rozpocząć proces synchronizacji danych wcześniej umieszczonych w bazie po stronie klienta. Jeśli klient ma dostęp do internetu wywoływane są funkcję, jak np. "synchronizeDecks".

```
1  self.synchronizeDecks = function (successCallback) {
2      var decks = self.decksToSync.get();
3
4      var url = "/Synchronize/SynchronizeDecks";
5      $.post(url, { json: htmlEncode(JSON.stringify(decks)) },
6                          function (data, textStatus) {
7          if (data.success) {
8              successCallback();
9              self.decksToSync.clean();
10         }
11         else
12             self.setNetworkStatus();
13     });
14 };
15
16 function LocalStorageArray(id) {
17     var self = this;
18
19     self.clean = function () {
20         localStorage[id] = JSON.stringify([]);
21     };
22
23     self.push = function (o) {
24         var array = JSON.parse(localStorage[id]);
25         array.push(o);
26         localStorage[id] = JSON.stringify(array);
27     };
28
29     self.get = function () {
30         var array = JSON.parse(localStorage[id]);
31         return array;
32     };
33
34     if(!localStorage[id])
35         self.clean();
36 };
```

W linii 2 następuje pobranie danych wcześniej zapisanych do bazy IndexedDB, ale nie zsynchronizowanych z bazą po stronie serwerowej. Tego typu dane postanowiono zapisać za pomocą klasy opakowującej(ang. wrapper) funkcjonalność LocalStorage o nazwie LocalStorageArray. Dzięki tej klasie możemy korzystać z danych zapisanych w LocalStorage w sposób bardzo podobny do zwykłej tablicy w języku JavaScript. Zaletą takiego rozwiązania jest zachowanie danych pomiędzy kolejnymi odwiedzeniami strony, podobnie jak w przypadku IndexedDB. Jest to niezbędne, ponieważ nie ma pewności czy klient połączy się z internetem przed wyłączniem przeglądarki internetowej po skończonej nauce. Dane w obiekcie LocalStorageArray o nazwie "decks" są ustawione przez obiekt

"repository" w momencie zapisu do IndexedDB, a więc dane znajdują się jednocześnie w LocalStorage i IndexedDB do czasu synchronizacji. Po synchronizacji są LocalStorageArray jest czyszczone.

W zmiennej "url" w lini 4 znajduje się adres do metody kontrolera ASP.NET. Metoda ta przyjmuje dane w formacie JSON(ang. JavaScript Object Notation), które należy dodać lub uaktualnić w bazie serwerowej i wygląda następująco:

```
1  [HttpPost]
2  public JsonResult SynchronizeDecks(FormCollection formCollection)
3  {
4      string decksJson = formCollection["json"];
5      decksJson = HttpUtility.HtmlDecode(decksJson);
6
7      var decks = JsonConvert.DeserializeObject<List<Deck>>(decksJson);
8
9      try
10     {
11         foreach (var deck in decks)
12         {
13             var deckToSync = _db.Decks.FirstOrDefault(d =>
14                 d.UserId == WebSecurity.CurrentUserId
15                 && d.ClientId == deck.ClientId);
16
17             if (deckToSync == null)
18             {
19                 deck.UserId = WebSecurity.CurrentUserId;
20                 _db.Decks.Add(deck);
21             }
22             else
23             {
24                 deckToSync.UserId = WebSecurity.CurrentUserId;
25                 deckToSync.Title = deck.Title;
26             }
27         }
28
29         _db.SaveChanges();
30     }
31     catch (Exception ex)
32     {
33         return Json(new { success = false });
34     }
35
36     return Json(new { success = true });
37 }
```

W linii 4 i 5 pobieramy z kolekcji wartości wysłanego formularza dane do synchronizacji i dekodujemy je. Następnie z formatu JSON wykonujemy deserializację, czyli przekształcenie danych z postaci tekstowej do struktury listy w języku C#. Następnie iterujemy po

elementach tej listy. W lini 13 za pomocą zapytania LINQ(ang. Language INtegrated Query) odwołujemy się do bazy. Zapytanie to jest zamienione na język SQL dzięki technologii EF(ang. Entity Framework). Próbujemy pobieramy w nim rekord przepisany do danego użytkownika i o danym id w bazie po stronie klienta. W ten sposób w dalszej części kodu możemy określić czy dane należy dodać - brak rekordu w bazie czy uaktualinić - rekord znajduje się już w bazie. W linii 29 zapisujemy zmiany. Całość wykonuje się w bloku try-catch. W przypadku niepowodzenia zwracamy obiekt w notacji JSON z flagą sygnalizującą niepowodzenie.

4.5. Generowanie manifestu AppCache

By możliwa była praca aplikacji w trybie bez dostępu do internetu należy utworzyć plik, która zawiera w sobie wszystkie pliki na serwerze, z których składa się aplikacja. Są to pliki o rozszerzeniach html, js, css oraz obrazki, stanowiące szablon graficzny.

Nazwy plików o rozszerzeniu html, css i js są łączone automatycznie dla oszczędzenia liczby żądań do serwera. Dodawany jest również ContentHash z powodów opisanych w rozdziale "Visual Studio i ASP.NET jako środowisko pracy". W tej sytuacji każda zmiana treści pliku zmusza programistę do zmiany manifestu AppCache. Dla celów automatyzacji tej czynności użyto technologii ASP.NET - "ASHX Handler", dzięki niej pod adresem "/Handlers/Manifest.ashx" przeglądarka internetowa pobierze manifest mimo, że nie jest on fizycznie dostępny na dysku serwera. Treść zawarta w manifeście jest wygenerowana dynamicznie, a następnie zapisana, dla zwiększenia wydajności w pamięci cache.

4.6. Testy

Aplikacja została przetestowana na komputerze z systemem Windows 8 na przeglądarkach internetowych Google Chrome 32 oraz Internet Explorer 11. Testy przeprowadzono również na smartphonie Samsung Galaxy S4 z system Android 4.3 na przeglądarce internetowej Google Chrome 32.

Scenariusze testowe:

Scenariusz 1

1. Uruchomienie aplikacji na przeglądarce z wyczyszczonymi danymi (ciasteczka, historia, LocalStorage, IndexedDb) z dostępem do Internetu.
2. Założenie nowego konta.
3. Wprowadzenie trzech nowych talii.
4. Wprowadzenie nowych informacji do każdej z talii.
5. Na ekranie z listą talii powinny znajdować się trzy pozycję.

6. Rozpoczęcie nauki informacji z pierwszej talii.
7. Przejście wszystkich informacji.
8. Powrót do listy talii. Powinny znajdować się dwie pozycje.

Scenariusz 2

1. Uruchomienie aplikacji następnego dnia na przeglądarce ze scenariusza 2 bez dostępu do Internetu.
2. Na liście talii powinny wyświetlać się trzy talie z informacjami do powtórki.
3. Wykonanie powtórki wszystkich informacji.
4. Na liście talii nie powinny wyświetlać się żadne pozycje.
5. Podłączenie dostępu do Internetu.
6. Powinna nastąpić synchronizacja danych z serwerem.

Scenariusz 3

1. Uruchomienie aplikacji na przeglądarce z wyczyszczonymi danymi z dostępem do Internetu.
2. Zalogowanie na konto, które posiada zsynchronizowane z serwerem dane.
3. Po zalogowaniu powinny wyświetlić się wcześniej zsynchronizowane dane.

Przeprowadzono również testy w których włączano i wyłączano dostęp do Internetu podczas dodawania nowego materiału oraz podczas powtórek.

Rozdział 5

Podsumowanie

Celem niniejszej pracy było opracowanie aplikacji, która implementuje szeroko wykorzystywany algorytm SuperMemo w wersji drugiej. Zrealizowane zostały założone funkcjonalności oraz nowoczesna forma. Wykorzystane zostały wszelkie mechanizmy dostępne w nowoczesnych przeglądarkach internetowych.

Funkcjonalość do wdrożenia w przyszłości to import i eksport danych z programów konkurencyjnych, przede wszystkim aplikacji Anki. Pozwoli to potencjalnym użytkownikom na podjęcie decyzji, która aplikacja w większym stopniu spełnia ich oczekiwania.

Bibliografia

[1] Benjamin LaGrone, *HTML5 and CSS3 Responsive Web Design Cookbook* – PACKT Publishing, 2013

[2] Bruce Johnson, *Professional Visual Studio 2012* – Wrox, 2012

[3] Adam Freeman, *Pro ASP.NET MVC 4* – Apress, 2012, rozdział 24

[4] Andrew Thomassen, *Learning and Forgetting Curves : A Practical Study* – University of Waikato

[5] Piotr A. Wozniak, *Optimization of learning* – Master's Thesis

[6] http://www.supermemo.com/english/history.htm

[7] http://www.supermemo.com/english/ol/sm2.htm

[8] http://zajaczkowski.org/2008/11/01/powtarzaj-i-odtwarzaj/

Spis rysunków

Załączniki

1. Płyta CD z niniejszą pracą w wersji elektronicznej.
2. Oświadczenie o oryginalności pracy i możliwości jej wykorzystania.