Tomasz PAŁYS

Uniwersytet Szczeciński, Wydział Matematyczno-Fizyczny, Katedra Edukacji Informatycznej i Technicznej *(absolwent)*
University of Szczecin, Faculty of Mathematics and Physics, Department of Informatics and Technical Education *(graduate)*

PROJEKTOWANIE INTERAKTYWNEGO INTERFEJSU GRAFICZNEGO STRONY INTERNETOWEJ Z WYKORZYSTANIEM BIBLIOTEKI JQUERY

Streszczenie

Wstęp i cele: W pracy przedstawiono opis biblioteki *jQuery* oraz jej możliwości. Ponadto praca opisuje tworzenie „krok po kroku" interaktywnej zawartości strony WWW z wykorzystaniem biblioteki *jQuery*. Głównym celem opracowania jest przedstawienie algorytmu tworzenia strony WWW z użyciem języka programowania *JavaScript.*
Materiał i metody: W pracy wykorzystano własny projekt tworzenia strony internetowej. Zastosowano programowanie w języku *JavaScript.*
Wyniki: W projektowaniu przestawiono pełny algorytm pisania własnej strony WWW z użyciem języka programowania *JavaScript.*
Wnioski: Standardy takie jak HTML5 i CSS3 nie są jeszcze w pełni wspierane przez twórców topowych przeglądarek internetowych. Wymusza to niekiedy wprowadzanie do stron dodatkowych linijek kodu (zazwyczaj w plikach CSS) optymalizujących je kod kątem konkretnej przeglądarki. Wydłuża to czas projektowania strony i zmniejsza przejrzystość kodu.

Słowa kluczowe: Programowanie, język *JavaScript,* strona internetowa.
(Otrzymano: 10.02.2015; Zrecenzowano: 15.02.2015; Zaakceptowano: 25.03.2015)

DESIGN OF INTERACTIVE GRAPHIC USER INTERFACE OF WEBSITE USING THE JQUERY LIBRARY

Abstract

Introduction and aims: *The paper presents a description of the jQuery library and its capabilities. In addition, the work describes some creation the method "step by step" interactive website by using jQuery library. The main aim of this paper is to present an algorithm creating website using JavaScript programming language.*
Material and methods: *The study was based on own design creating a website. It has been used JavaScript program.*
Results: *In presented design has been shown the full algorithm of writing own website using the JavaScript programming language.*
Conclusions: *Standards such as HTML5 and CSS3 are not yet fully supported by the creators of the top web browsers. This forces the parties sometimes placing additional lines of code (usually in the CSS files) to optimize them for specific browser code. This prolongs the site design and reduces the transparency of the code.*

Keywords: *Programming language JavaScript, website.*
(Received: 10.02.2015; Revised: 15.02.2015; Accepted: 25.03.2015)

1. Wstęp

Początkowo twórcy stron internetowych mieli do dyspozycji tylko jeden język ich programowania. Był to HTML, a właściwie jego szkic. Język ten był bardzo ubogi, wskutek czego strony były pisane bardzo prosto i statycznie. Wraz z wprowadzeniem oficjalnego standardu HTML 2.0 (1996 r.) powstał również język CSS (Kaskadowe arkusze stylów, poziom 1), który systematyzował formatownie wyglądu graficznego strony. Umożliwiło to uzyskanie identycznej prezentacji stron w różnych przeglądarkach. W drugiej połowie lat dziewięćdziesiątych pojawił się też język skryptowy *JavaScript.* Rozwój tych technologi (oraz pojawianie się innych) umożliwiło tworzenie coraz bardziej rozbudowanych, atrakcyjnych wizualnie, posiadających animowane i interaktywne elementy stron WWW.

Projekt miał na celu stworzenie strony internetowej korzystając z najnowszych wzorców i standardów projektowania oraz technologii, takich jak HTML5, CSS3. Podczas projektowania strony wykorzystano język programowania *JavaScript.* Szczególny nacisk położono na użycie biblioteki programistycznej *jQuery* (nakładka na *JavaScript*). Programy jakie zostały użyte do tworzenia strony to Geany[1] i GIMP[2].

2. Biblioteka jQuery

Dzięki zastosowaniu stylów CSS można tworzyć efektowne wizualnie strony. Niestety obecnie ładny wygląd to za mało. Strona musi żyć, wchodzić w interakcję z urzytkownikiem. Tu z pomocą przychodzi biblioteka jQuery i ogrom jej możliwości. Interaktywne formularze i dynamiczne galerie zdjęć, efektowene menu kontekstowe, pojawiające się lub znikające elementy strony. Wymienione efekty jak i wiele innych możemy osiągnąć załączając bilbiotekę jQuery do naszej strony internetowej.

2.1. Czym jest jQuery?

JavaScript (w skrócie JS) jest to skryptowy język programowania wykorzystywany na stronie internetowej w celu umożliwienia modyfikacji jej zawartości w czasie rzeczywistym poprzez reagowanie na zdarzenia (np. klikniecie, najechanie myszką na jakiś obiekt strony). Niestety pisanie skryptów w tym języku jest czasochłonne. Niezbyt skomplikowane efekty, zdarzenia wymagają dużej ilości kodu do ich opisania co powoduje, że skrypty stają się skomplikowane i mało czytelne.

Biblioteka *jQuery* powstała, aby rozwiązać wyżej wymienione problemy. Została napisana w *JavaScript* i zawiera w sobie metody opisujące typowe zdarzenia, które możemy wywołać za pomocą jednej linii kodu. Dodatkowo upraszcza wybieranie elementów HTML dzięki zastosowaniu selektorów bazujących na CSS3.

Poniżej przedstawiona jest funkcja ukrywająca[3] element strony po kliknięciu na niego, napisana jest w czystym JS:

```
var el = document.getElementById('myElement');
el.onclick = function(e) {
    fadeOut(750, this);
}
function fadeOut(ms, el) {
    var opacity = 1,
```

[1] Darmowy edytor tekstu zawierający podstawowe cechy zintegrowanego środowiska programistycznego wspierający wiele języków programowania *(http://www.geany.org/).*

[2] Darmowy program do tworzenia i obróbki grafiki *(http://www.gimp.org/).*

[3] Kod zaczerpnięty ze strony: http://toddmotto.com/raw-javascript-jquery-style-fadein-fadeout-functions-hugo-giraudel/, *(dostęp 27.01.2013).*

```
        interval = 50,
        gap = interval / ms;

    function func() {
        opacity -= gap;
        el.style.opacity = opacity;

        if(opacity <= 0) {
            window.clearInterval(fading);
            el.style.display = 'none';
        }
    }
    var fading = window.setInterval(func, interval);
}
```

Powyższy skrypt z wykorzystaniem biblioteki *jQuery* wygląda następująco:

```
$('myElement').fadeOut(750);
```

Jak widać *jQuery* ułatwia pisanie w JavaScript pozwalając zaoszczędzić czas. Kod staje się o wiele prostszy i bardziej przejrzysty.

2.2. Implementacja biblioteki jQuery na stronie WWW

Zanim zaczniemy korzystać z dobrodziejstw biblioteki *jQuery* musimy ją dodać do strony. Najnowszą wersję biblioteki można pobrać ze strony projektu (**www.jquery.com**). Plik dodajemy tak samo jak to miało miejsce w przypadku kaskadowych arkuszy stylów w sekcji **head** dokumentu HTML wykorzystując w tym celu znacznik <**script**>:

```
<head>
   <script src="js/jquery-1.9.1.min.js"></script>
</head>
```

W atrybucie **src** podaje się ścieżkę do pliku z biblioteką. Istnieje też druga możliwość implementacji jQuery nie wymagająca ściągania biblioteki bezpośrednio na nasz dysk twardy. Korzystając z tak zwanej sieci CDN[4] *(ang. Content Delivery Network)* w atrybucie **src** można wpisać link internetowy do pliku z najnowszą dostępną wersją biblioteki:

```
<head>
  <script src="http://code.jquery.com/jquery-latest.min.js"></script>
</head>
```

Korzystając z CDN trzeba mieć na uwadze fakt, że wraz z pojawieniem się nowej wersji jQuery skrypty napisane w starszej mogą przestać działać i będą wymagały modyfikacji.

Gdy biblioteka jest już wgrana przystępujemy do tworzenia własnych skryptów, które możemy pisać w zewnętrznym pliku i załączyć do strony tak jak plik biblioteki jQuery. Jeśli skrypty nie zajmują dużo miejsca (mało linijek kodu) można umieścić je bezpośrednio na stronie w sekcji **head**:

```
<head>
   <script>
     Nasze krypty
   </script>
</head>
```

[4] System serwerów, którego zadaniem jest szybkie i wydajne dostarczanie zawartości Internetu do użytkownika końcowego

2.3. Składnia w jQuery

Pisanie skryptów w *jQuery* jest bardzo proste. W skrócie wygląda to tak: wywołujemy bibliotekę, wybieramy elementy HTML i wykonujemy na nich określone działania. Skrypty mają następującą składnię:

```
jQuery('selektor').akcja(); lub  $('selektor').akcja();
```

Wywołanie biblioteki następuje poprzez zastosowanie napisu **jQuery** lub znaku **$**. Jeśli często odwołujemy się do biblioteki *jQuery* znak dolara skraca pisanie i poprawia czytelność skryptu. Trzeba jednak zwrócić uwagę czy nie stosujemy innych bibliotek, które również korzystają z tego znaku. Po wywołaniu biblioteki przyszedł czas na wybranie elementów. Selektory stosowane w *jQuery* bazują na tych znanych z kaskadowych arkuszy stylów. Jeśli chcemy wybrać więcej różnych obiektów HTML selektory oddzielamy znakiem przecinka:

```
$('img, a, p, .klasa').akcja();
```

Akcje poprzedzamy znakiem kropki. Określają one jakie zdarzenia i efekty zostaną przeprowadzone na wybranych elementach. Możemy łączyć je w łańcuchy. Tworzy się kolejka działań, które będą wykonywane jedno po drugim na danych elementach:

```
$('img').akcja1().akcja2().akcja3().akcja4();
```

Efekty i zdarzenia mogą opcjonalnie posiadać parametry, które je definiują. Każdej akcji możemy przypisać sekwencję kodu, który zostanie wykonany po jej zajściu:

```
$('selektor').akcja('parametr1','parametr2', function() {
  Kod, który zostanie wykonany po zajściu akcji
});
```

Na poniższym przykładzie widać, że kliknięcie w element h1 powoduje wywołanie kodu zmieniającego kolor na czerwony wszystkich akapitów na stronie:

```
$('h1').click( function() {
  $('p').css('color', 'red');
});
```

2.4. Zdarzenia

Wszystkie różne akcje użytkownika (np. kliknięcie w link, najechanie myszką na jakiś element strony), na które strona jest w stanie odpowiedzieć określamy mianem zdarzenia. Zanim skrypty zaczną działać dokument HTML musi zostać przetworzony przez przeglądarkę w celu stworzenia szkieletu strony. Znaczniki są odczytywane i na ich podstawie przygotowywane są elementy HTML.

Aby zapobiec wykonywanie się skryptu na nieistniejących jeszcze elementach stosuję się instrukcję **$(document).ready**, która wykona wewnętrzny kod dopiero po przygotowaniu wszystkich elementów przez przeglądarkę.

```
$(document).ready( function() {
    ...
      Tutaj znajdują się skrypty jQuery
...
});
```

Przygotowanie elementów nie oznacza ich pełnego wczytania. Będzie tylko wiadomo, że na stronie znajduje się element taki jak np. grafika (znacznik <img>), jednak jego zawartość jeszcze nie została wczytana. Jeśli chcemy aby skrypt został wykonany po pełnym

załadowaniu strony (element **img** będzie miał załadowaną zawartość i będzie ona wyświetlona w przeglądarce) musimy skorzystać z instrukcji **$(window).load**.

```
$(window).load( function() {
     ...
       Tutaj znajdują się skrypty jQuery
...
});
```

Po przeanalizowaniu zachowania się tych dwóch zdarzeń widać, że skrypty zawarte w $(document).ready uruchamiają są wcześniej niż te z $(window).load. Jeśli skrypt ma zostać wykonany na elemencie, którego wczytanie może chwilę potrwać (zdjęcie sporych rozmiarów) to skrypt ten umieścimy w zdarzeniu $(window).load, aby wykonał się dopiero po załadowani i wyświetleniu zdjęcia na monitorze. Zdarzenia $(document).ready używamy np. w sytuacji, gdy chcemy ukryć dany element i zapobiec jego wyświetleniu na ekranie.

2.5. Manipulowanie elementami HTML

Biblioteka *jQuery* jest potężnym narzędziem pozwalającym w łatwy sposób „bawić" się elementami dokumentu HTML. W podrozdziale tym dowiemy się jak stosować różne efekty, dodać/usunąć zawartość strony. Zostanie omówione manipulowanie atrybutami, stylami CSS oraz wymiarami elementów.

♦ **Efekty**

Z *jQuery* w łatwy sposób można ukrywać lub pokazywać wcześniej ukryte elementy HTML. Aby ukryć element stosuje się instrukcje **hide()**. W celu pokazania elementu używa się metodę **show()**. Obydwie instrukcje można wywołać z parametrem, który charakteryzuje czas (wpisany w milisekundach lub jako slow, fast) ukrywania/pokazywania elementu. Bez podawania tego parametru czas domyślnie równy jest zeru, elementy są ukrywane/pokazywane natychmiast.

```
$('img').hide(1000);
$('img').show('slow');
```

Innym sposobem pokazywania, ukrywania elementów jest stosowanie instrukcji **fadeOut()**, **fadeIn()**. Od poprzednich metod odróżnia je sposób w jaki chowane/pokazywane są obiekty na stronie. Elementy są rozjaśniane, ściemniane tzn. element jest animowany od stanu maksymalnej przezroczystości (niewidoczny) do momentu 100% widzialności (element nieprzezroczysty). Oprócz parametru czasu można podać dodatkowy charakteryzujący przebieg tych efektów.

```
$('#foto').fadeIn(500, 'easeInQuad');
```

Istnieje też instrukcja **fadeTo()**, którą możemy ustalić przezroczystości danego elementu. W parametrach wpisuje się czas animacji, poziom widzialności oraz opcjonalnie parametr charakteryzujący przebieg animacji.

```
$('figure').fadeTo(500, 0.5, 'linear');
```

Każdy efekt, zanim zostanie skończony, można zatrzymać w dowolnej chwili stosując instrukcję **stop()**. Metodę tą możemy wywołać z dwoma parametrami przyjmującymi wartości true albo false. Pierwszy parametr mówi czy chcemy zatrzymać tyko aktualnie wykonywany efekt (false) czy wszystkie następne (true) występujące po nim na wybranym elemencie. Drugi określa czy dokończyć natychmiast (bardzo szybko), czy też nie, aktualną animację.

♦ **Style CSS**

Biblioteka *jQuery* pozwana na manipulowanie kaskadowymi arkuszami stylów. Korzystając z instrukcji **css()** można odczytać albo ustawić właściwości stylu elementu HTML. Jeśli chcemy pozyskać aktualny styl elementu tworzymy zmienną, do której przypiszemy wartość odczytanego stylu:

```
var nazwa_zmiennej = $('selektor').css('nazwa_właściwości_stylu');
var rozmiar_tekstu = $('p.duzy_tekst').css('font-size');
```

Wartość utworzonej zmiennej rozmiar_teksu, która przechowuje rozmiar czcionki możemy przypisać innemu elementowi. W tym celu stosujemy instrukcję css() z dodatkowym parametrem, który określi wartość stylu:

```
$('selektor').css('właściwość_stylu','wartość');
$('p').css('font-size', rozmiar_tekstu);
```

Można za jednym razem przypisać więcej definicji stylów do wybranego elementu korzystając z następującej składni:

```
$('selektor').css({'właściwość':'wartość,'właściwość1':'wartość1'...});
```

Dla lepszej czytelności kodu dobrym pomysłem jest pisanie każdej deklaracji w nowej linii.

```
$('p').css({
  'font-size':'20px',
  'color':'red',
  'text-align':'center',
  'background-color':'yellow'
});
```

♦**Atrybuty**

Do manipulowania atrybutami stosuje się instrukcję **attr()**. Pozwala ona odczytać albo ustawić wartość wybranego atrybutu elementu HTML. Jeśli chcemy odczytać wartość atrybutu korzystamy z następującego kodu:

```
$('selektor').attr('atrybut');
$('img').attr('src');
```

Wartość odczytanego atrybutu można przypisać do zmiennej i wykorzystać w dalszej części pisanego skryptu. W celu zmiany wartości atrybutu stosujemy tą samą instrukcję z dodatkowym parametrem określającym wartość atrybutu:

```
$('selektor').attr('atrybut','wartość');
$('a').css('href','http://www.google.pl');
```

Biblioteka *jQuery* posiada również instrukcję **val()**, która bezpośrednio odczytuje wartość atrybutu **value** (tekst wpisany przez użytkownika) w znaczniku **<input>** (wykorzystywany w formularzach do wprowadzania danych). Aby zmienić atrybutu **value** do instrukcji val() dodajemy parametrem z wartością jaka ma być do niego przypisana:

```
$('input#nazwisko').val('Twoje nazwisko');
```

Do manipulowania atrybutem **class** przeznaczone są instrukcje **addClass()** i **removeClass()**, które odpowiednio dodają i usuwają podaną klasę:

```
$('p').addClass('akapit');
$('p').removeClass('akapit');
```

♦ **Wymiary elementów**

Biblioteka *jQuery* posiada kilka metod do pracy z wymiarami elementów. Można ich używać do pobierania aktualnej szerokości/wysokości elementu jak również do

manipulowania nimi. Instrukcja bez wpisania parametru zwraca wymiar elementu w pikselach. Wyróżnia się następujące instrukcje sterujące wymiarami:

– width / height()
– innerWidth / Height()
– outerWidth / Height()
– outerWidth / Height(true)

♦ Dodawanie/usuwanie

Dodawanie nowych elementów jak i zawartości do już istniejących jest bardzo proste. Służą do tego następujące instrukcje:

– append(), wstawia zawartość na końcu wybranych elementów
– prepend(), wstawia zawartość na początku wybranych elementów
– after(), wstawia zawartość po wybranych elementach
– before(), wstawia zawartość przed wybranymi elementami

Przykładowo chcąc dodać tekst na końcu pierwszego akapitu skorzystamy z metody **append()** w parametrze wpisując tekst, który chcemy dodać.

```
$(p:first').append('Ten tekst zostanie wyświetlony na dole akapitu');
```

Jeśli chcemy wstawić nowy akapit przed już istniejącym użyjemy instrukcji **before()** w parametrze wpisując znaczniki tworzące ten element HTML.

```
$(p:first').before('<p> Tekst utworzonego akapitu</p>');
```

Chcąc wydobyć tekst z elementu stosuje się instrukcję **text()**. Można nią też zastąpić cały tekst w wybranym elemencie.

```
$(p:first').text('Nowy tekst wyświetlany w akapicie');
```

Do usuwania elementów służy instrukcja **remove()**, która usuwa wybrany przez selektor element wraz z jego zawartością. Aby wyczyścić zawartość elementu nie usuwając go stosuje się metodę **empty()**.

2.6. Funkcje

Jak już wcześniej było wspomniane każdej akcji można przypisać kod wewnętrzny, który zostanie wywołany po zajściu danego zdarzenia, efektu. Kod ten to tzw. funkcja zwrotna *(ang. callback function).* Wszystkie nasze stypy są funkcją zwrotną dwóch podstawowych zdarzeń $(dokument)..ready() oraz $(window).load(), są w nich zawarte. Do funkcji zwrotnej można przypisać argument (zmienną), który będzie do niej przekazywany.

```
$(document).ready( function() {
$('a').click( function(zdarzenie) {
     zdarzenie.preventDefault();
});
   });
```

Argument **zdarzenie**, w którym zapisana jest domyśla akcja wybranego elementu (zmiana strony po kliknięciu w link) jest przekazywany do funkcji zwrotnej, gdzie wykorzystując instrukcje **zdarzenie.preventDefault()** zatrzymuje się domyślną akcję znacznika **<a>**. Jeśli w funkcji zwrotnej chcemy odwołać się do elementu ją wywołującego stosujemy selektor **this**.

```
$('section figure:not(.img_1, .img_2)').click(function(){
  $(this).fadeTo(350, 0.5);
});
```

JavaScript pozwala również na tworzenie własnych funkcji, które można wywoływać z dowolnego miejscu w kodzie. Funkcje te mogą posiadać argumenty (zmienne), które będą do nich przekazywane i wykorzystywane wewnątrz funkcji.

```
function moja_funkcja(zmienna1, zmienna2)
{
Kod wykonywany przez funkcję
  }
```

3. Projektowanie strony z wykorzystaniem jQuery

Rozdział jest poświecony opisowi krok po kroku jak z wykożystaniem biblioteki jQuery twożyć interaktywną zawartość strony internetowej. Przedstawiony zostanie sbosób na ukrycie wczytywania się strony, jak również efekt przejścia między stronami. Omówione zostaną składowe strony takie jak: zmieniające się tło strony, formularz kontaktowy z weryfikacją danych, galeria zdjęć, wyszukiwarka zawartości strony z automatycznie wyświetlanymi podpowiedziami.

3.1. Ładowanie strony, przejście między stronami

Załóżmy, że na stronę wchodzi użytkownik, który ma wolne łącze internetowe lub serwer, na którym znajduje się strona jest obciążony powodując spowolnienie wysyłania danych do użytkowników. Jeśli strona posiada dużo treści graficznych w wysokiej rozdzielczości na ekranie można będzie zaobserwować wczytywanie się poszczególnych fragmentów strony. Aby temu zapobiec strona wraz z wszystkimi jej elementami powinna być wczytana i dopiero wtedy wyświetlona użytkownikowi. W podrozdziale tym zostanie omówiona konstrukcja i schemat budowy krok po kroku prostego mechanizmu ładowania strony i wyświetlenia jej dopiero po całkowitym wczytaniu.

Pierwszym krokiem jest ukrycie elementów strony aby były niewidoczne. W tym celu skorzystamy z kaskadowych arkuszy stylów ustawiając właściwość **display: none**. Następnie tworzymy elementy HTML, które będą informować użytkownika o wczytywaniu.

```
<div id="wczytywanie">
<div id="loading_img">
<img src="img/loading.gif">
</div>
<div id="loading_text">
<h2>WCZYTYWANIE...</h2>
<h3>Proszę czekać.</h3>
</div>
</div>
```

Elementowi z identyfikatorem **id=”wczytywanie”** również ustawiamy w CSS **display: none**. Teraz przyszedł czas na użycie jQuery. W funkcji zwrotnej zdarzenia **ready()** korzystamy z efektu **fadeIn()** i wyświetlamy element **wczytywanie** wraz z jego zawartością.

```
$(document).ready(function() {
$('#wczytywanie').fadeIn(1500, 'easeInQuad');
  });
```

Właściwa zawartość strony cały czas jest ukryta i wczytuje się. Aby ją wyświetlić w funkcji zwrotnej zdarzenie **load()** ukrywamy element **wczytywanie** następnie wyświetlamy całkowicie załadowaną zawartość strony.

```
$(window).load(function() {
$('#wczytywanie').stop(true,false).hide();
$('.tlo_biale, #tlo').fadeIn(500, 'easeInQuad');
```

```
});
```

Efekt przejścia między stronami uzyskujemy dodając w funkcji zwrotnej zdarzenia **load()** następujący kod.

```
1  $("a.przejscie").click(function(zdarzenie){
2     zdarzenie.preventDefault();
3     var link_do_strony = $(this).attr('href');
4     $(".tlo_biale, #tlo").fadeOut(400, 'easeInQuad', function(){
5       window.location = link_do_strony;
6     });
7  });
```

Klikając w link do funkcji zwrotnej przekazywana jest domyślna akcja tego zdarzenia (linia pierwsza kodu) i tam zatrzymywana (linia druga). Następnie do zmiennej **link_do_strony** przypisywany jest adres strony (linia trzecia), która ma się wczytać. W czwartej linii wywołujemy efekt animacji ukrywania aktualnej zawartości strony. W wierszu piątym korzystamy z instrukcji **window.location**, która mówi przeglądarce jaka strona ma być wczytana.

3.2. Zmieniające się tło strony

W kodzie strony stosując znaczniki ** ** tworzymy pusty element HTML z identyfikatorem **id=”tlo”**. Jest on pusty, ponieważ obraz tła wstawimy poprzez odpowiednie deklaracje stylów w pliku CSS.

```
1  #tlo {
2     display: none;
3     position: fixed;
4     left: 0;
5     top: 0;
6     width: 100%;
7     height: 100%;
8     background-image: url(../img/background/bg1.jpg);
9     background-position: center;
10    background-repeat: no-repeat;
11    background-size: cover;
12       }
```

Element **tlo** jest pozycjonowany względem okna przeglądarki (3)[5] w lewym górnym rogu (4,5) i rozciągnięty na całej jej długości (6), wysokości (7). Wstawiony do elementu obraz tła (8) jest wyśrodkowany (9) i nie powtarza się (10). Wysokość i szerokość wstawionego obrazu jest dopasowywana tak aby całkowicie pokrył element (11).

Mając element HTML wyświetlający tło możemy napisać skrypt, który będzie je zmieniał. Chcemy aby tapeta zaczęła się zmieniać dopiero po załadowaniu i wyświetleniu strony, więc skrypt umieścimy w funkcji zwrotnej zdarzenia $(window).load(). Najpierw tworzymy zmienną **tapeta** (1), która jest tablicą jednowymiarową. Jej elementami są nazwy plików tła strony. Każdy element ma przypisany do siebie indeks, aby w łatwy sposób można go było później odczytać. Tablice indeksowane są od zera. Obiekt **tapeta** posiada pięć elementów o indeksach: pierwszy – 0, drugi – 1, trzeci – 2, czwarty – 3, piąty – 4. Definiujemy drugą zmienną o nazwie **numer** (2), która będzie wskazywać indeks elementu w tablicy **tapeta**. Korzystając z wbudowanej funkcji JavaScript **setInterval(kod, czas)** wywoływać będziemy kod zmieniający obraz tła co określony odstęp czasu (3). Kod zmieniający tło zapiszemy w funkcji **zmiana_tla** (5-11). Zmienna **numer** za każdym wywołaniem funkcji musi

[5] Numer wiersza w kodzie.

wskazywać na kolejny element z tablicy. Stosując odpowiedni wzór (7)[6] zmienna ta zawsze będzie przyjmować wartość od zera do czterech (nasze indeksy elementów w tablicy) to jest: 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0… itd. Aktualne tło ukrywamy (8) poczym zmieniamy wartość deklaracji stylu **background-image,** gdzie w miejsce nazwy pliku wstawiamy element z tablicy **tapeta** o indeksie wskazanym przez zmienną **numer** (9). Tło ze zmienionym obrazem wyświetlamy na ekranie (10).

```
1  var tapeta = ["bg1.jpg","bg2.jpg","bg3.jpg","bg4.jpg","bg5.jpg"];
2  var numer = 0;
3  setInterval( zmiana_tla, 7000);
4
5  function zmiana_tla()
6  {
7   numer = (numer + 1) % tapeta.length;
8   $('#tlo').fadeOut(400, 'easeInSine' ,function(){
9      $(this).css({'backgroundImage':'url(img/background/'+tapeta[numer]+')'});
10  }).fadeIn(800, 'easeOutSine');
11 }
```

3.3. Galeria zdjęć

Galeria składa się z szeregu miniaturek zdjęć wyświetlanych na ekranie obok siebie. Po najechaniu kursorem na miniaturkę wyświetli się na niej ikona (lupa) informująca użytkownika o możliwości wyświetlenia zdjęcia w powiększeniu. Po kliknięciu w miniaturkę strona zostanie przyciemniona dzięki nakładce (czarne transparentne tło), która uniemożliwi interakcję użytkownika z innymi elementami strony. Na środku okna przeglądarki wyświetli się zdjęcie w oryginalnej rozdzielczości. Kliknięcie w zdjęcie spowoduje schowanie go i nakładki umożliwiając dalszą interakcję ze stroną.

Pojedynczy elementy miniatury zdjęcia tworzymy poniższym kodem HTML odpowiednio go formatując przy użyciu kaskadowych arkuszy stylów.

```
  ...
<figure>
         <div class="zdjecie">
          <img src="img/gallery/galeria1/zdj1.jpg">
          <img class="gallery_big" src="img/gallery_big/galeria1/zdj1.jpg">
          <span class="zoom" title="zoom"></span>
         </div>
</figure>
...
```

Transparentną nakładkę (znacznik ** ** z **id=”Overlay”**) na stronę jak i element, w którym będzie wyświetlane zdjęcie w oryginalnej rozdzielczości tworzy następujący kod:

```
<div id="Overlay"></div>
<div id="zoom-zjdecie">
<img id="foto_big" src="">
</div>
```

Obu elementom deklarujemy styl **display: none**. Będą wyświetlane dopiero po kliknięciu w miniaturę zdjęcia.

Mając gotowe elementy HTML można przystąpić do pisania skryptu w jQuery. Po kliknięciu w miniaturę zdjęcia (1) zostają wywołane następujące zdarzenia:

– ukrywany jest scroll uniemożliwiając przewijanie strony (2)
– do zmiennej **obrazek** przypisywany jest link do zdjęcia w normalnej rozdzielczości (3)
– do elementu wyświetlającego zdjęcie wstawiamy link ze zmiennej **obrazek** (4)

[6] Wynikiem równania jest reszta z dzielenia zmiennej **numer** zwiększonej o jeden przez ilość elementów w tablicy.

– ustalana jest odległość od lewej krawędzi ekranu (5)
– ustalana jest odległość od górnej krawędzi ekranu (6)
– wartości zmiennych **od_lewej**, **od_gory** przypisywane są elementowi wyświetlającemu zdjęcie w celu jego wyśrodkowania na ekranie (7)
– zostaje wyświetlona nakładka i element ze zdjęciem (8)

```
1  $('.galeria figure .zdjecie').click(function(){
2        $('body').css({'overflow':'hidden'});
3        var obrazek = $(this).find('img.gallery_big').attr('src');
4        $('#foto_big').attr('src', obrazek);
5        var od_lewej = ($(window).width()-$('#zoom-zjdecie').outerWidth())/2;
6        var od_gory =  ($(window).height()-$('#zoom-zjdecie').outerHeight())/2;
7        $('#zoom-zjdecie').css({'left':od_lewej,'top':od_gory});
8        $('#Overlay, #zoom-zjdecie').fadeIn(400);
9  });
```

Duże zdjęcie wraz z nakładką chowamy następującym skryptem:

```
$('#zoom-zjdecie').click(function(){
   $('body').css({'overflow':'auto'});
   $('#Overlay, #zoom-zjdecie').fadeOut(300);
});
```

Klikając w zdjęcie przywracana jest możliwość przewijania strony i ukrywana jest nakładka, zdjęcie.

Do wyświetlenia ikony lupy na miniaturze zdjęcia użyjemy zdarzenia **hover()**. Zawiera ono dwa parametry. Są to funkcję zwrotne mówiące co ma się stać po najechaniu myszką na element oraz co, gdy kursor go opuści. Po wczytaniu strony ikona jest niewidoczna, całkowicie transparentna (1). Gdy kursor znajdzie się na miniaturze zdjęcia ikona jest animowana do pełnej widzialności (3). Jednocześnie sama miniatura staje się lekko przeźroczysta (4). Opuszczając kursorem obszar miniatury ta staje się nieprzeźroczysta (7), a ikona lupy znika (6).

```
1  $('.zoom').css({'opacity':0});
2  $('.zoom').hover(
3        function(){$(this).stop(true,true).animate({'opacity': 1 }, 'slow');
4        $(this).siblings('img').stop(true,true).animate({'opacity':0.8},250);},
5
6        function(){$(this).stop(true,true).animate({'opacity': 0 },'fast');
7        $(this).siblings('img').stop(true,true).animate({'opacity':1},'fast');
8  });
```

3.4. Wyszukiwarka zawartości strony

Wyszukiwarka działa na zasadzie wyświetlania odpowiedniej strony w zależności od podanego przez użytkownika wyrazu lub frazy. Składa się z dwóch skryptów. Pierwszy porównuje wpisywane znaki z wcześniej zdefiniowaną bazą danych i wyświetla pasujące wyrażenia. Drugi po kliknięciu w ikonę szukania wczytuje odpowiednią podstronę, jeżeli wpisana fraza została znaleziona w bazie danych. Gdy nie zostanie znaleziona użytkownik zostanie o tym poinformowany w wyświetlonym na ekranie komunikacie.

Kod HTML potrzebny do jego stworzenia widoczny jest poniżej. Oczywiście aby wyglądał tak jak na rysunku, musi zostać odpowiednio sformatowany przy użyciu kaskadowych arkuszy stylów.

```
...
<div id="search">
<form id="form1">
   <input class="znajdz" type="text" placeholder="Znajdź na stronie…">
   <a class="wyszukaj" target="_self"  href="#">
```

```
      <img src="img/szukaj2.png" class="szukaj">
    </a>
    <ul id="lista_szukane">
      <li>Film</li>
      <li>Filmy</li>
      <li>Wideo</li>
      <li>Video</li>
      <li>Zdjęcie</li>
      <li>Zdjęcia</li>
      <li>Fotografia</li>
      <li>Fotografie</li>
      <li>Kontakt</li>
      <li>Formularz kontaktowy</li>
      <li>Email kontaktowy</li>
      <li>Galeria</li>
      <li>Strona główna</li>
    </ul>
  </form>
  </div>
  ...
```

W znaczniku <input> został wykorzystany nowy atrybut wprowadzony w HTML5 **placeholder**. Pozwala on wyświetlać w polu tekstowym informację o oczekiwanej wartości, formacie danych jakie ma podać użytkownik. Lista z id=”lista_szukane” i wszystkie jej elementy są wstępnie ukryte.

Automatyczne dopasowywanie frazy do wpisywanych przez użytkownika liter odbywa się za każdym wciśnięciem klawisza (1). Wykonywana są wtedy następująca zdarzenia. Zmienna **szukana** przyjmuje wartość wpisanych liter (2). Sprawdzane jest czy użytkownik coś wpisał (3). Jeśli nie ma wpisanego tekstu lista jest ukrywana (5), gdy wpisany wyświetlana (9). Następnie wykorzystując instrukcję **each()** wywołujemy funkcję zwrotna dla każdego dopasowanego przez selektor elementu oddzielnie (10). Dla elementu listy sprawdzany jest warunek czy wzorzec, którym jest wpisany przez użytkownika tekst, występuje w tekście tego elementu (11). Jeśli tak element ten jest wyświetlany (13), w przeciwnym przypadku ukrywany (17).

```
1  $('.znajdz').on('keyup click',function(){
2    var szukana = $(this).val()
3    if ( szukana.length == 0)
4    {
5     $('#lista_szukane').stop(true,true).fadeOut();
6    }
7    else
8    {
9     $('#lista_szukane').stop(true,true).fadeIn();
10    $('#lista_szukane li').each(function(){
11     if (new RegExp('^' + szukana + '', 'i').test($(this).text())==true)
12     {
13           $(this).show(250);
14     }
15     else
16     {
17            $(this).hide(250);
18     }
19        });
20   }
21   });
```

Skrypt zmieniający stroną w zależności od wpisanej frazy wykonuje następujące czynności. Po kliknięciu w ikonę szukania wywoływana jest funkcja zwrotna (1). Do zmiennej **co_szukać** zapisywana jest wartość pola tekstowego (2). Jeśli użytkownik nic nie

wpisze uruchomi się funkcja **info** wyświetlająca odpowiedni komunikat (5). W przeciwnym przypadku korzystając z instrukcji **inArray()** wartość zmiennej **co_szukac** jest szukana w tablicy **baza_danych**. Gdy element zostanie znaleziony zwracany jest jego indeks i przypisywany do zmiennej **index** (9). W zależności od wartości zmiennej **index** wywoływana jest funkcja **zmien_strone**, która wczyta odpowiednią stronę. Jeśli element nie został znaleziony **index = −1** powodując wywołanie funkcji **info**, która wyświetli odpowiedni komunikat (31).

```
var baza_danych=['galeria','zdjęcia','zdjęcie','fotografia','fotografie',
    'film','wideo','video','kontakt','formularz kontaktowy',
    'email kontaktowy','dane kontaktowe','strona główna'];
1  $('.wyszukaj').click(function(){
2    var co_szukac = $('.znajdz').val();
3    if ( co_szukac.length == 0 )
4       {
5        info('Wpisz czego chcesz szukać!');
6       }
7    else
8       {
9        var index = $.inArray(co_szukac, baza_danych);
10        if ( index > -1 )
11      {
12       if (index < 5)
13          {
14           zmien_strone('oferta.html');
15          }
16       if (index > 4 && index < 9)
17          {
18        zmien_strone('video.html');
19          }
20       if (index > 8 && index < 13)
21          {
22        zmien_strone('kontakt.html');
23          }
24       if (index > 12)
25          {
26        zmien_strone('index.html');
27          }
28      }
29        else
30      {
31       info('Nie znaleziono pasujących elementów!');
32      }
33       }
34  });
```

Do stworzenia wyświetlanie komunikatów wykorzystujemy nakładkę na ekran zrobioną na potrzeby galerii zdjęć dodając element, w którym będą wyświetlane komunikaty. Elementy HTML odpowiednio formatujemy w pliku CSS.

```
<div id="Overlay"></div>
<div class="info">
     <p></p>
     <div><button type="button" class="button">OK</button></div>
</div>
```

Funkcja **info** odpowiedzialna za wyświetlanie komunikatów bazuje na skrypcie wyświetlającym duże zdjęcia w galerii. Różni się tym, że zamiast wstawiać zdjęcie do wybranego elementu HTML korzysta z instrukcji **text()**, która zamienia tekst wskazanego elementu na podany w parametrze.

```
function info(informacja)
   {
    $('body').css({overflow:'hidden'});
      $('div.info p').text(informacja);
      var odlewej = ( $(window).width() - $('div.info').outerWidth() ) / 2;
      var odgory =  ( $(window).height() - $('div.info').outerHeight() ) / 2;
      $('div.info').css({'left':odlewej,'top':odgory});
      $('#Overlay, div.info').fadeIn(400);
     }
```

Skrypt zamykający przedstawia się następująco:

```
$('div.info .button').click(function(){
     $('body').css({overflow:'auto'});
     $('#Overlay, div.info').fadeOut(300);
});
```

Funkcja zmieniająca stronę jest bardzo prosta. Bazuje na wcześniej omawianym zagadnieniu przejścia między stronami.

```
function zmien_strone(page)
     {
      $('.tlo_biale, #tlo').fadeOut(400, 'easeInQuad', function(){
         window.location = page;
      });
   }
```

3.5. Formularz kontaktowy, weryfikacja danych

Weryfikacja danych wprowadzanych do formularza przez użytkownika odbywa się w czasie rzeczywistym. Po każdym wpisaniu nowego znaku, litery sprawdzana jest długość oraz poprawność danych. Gdy wprowadzone wartości są za krótkie lub nie pasują do wzorca użytkownik jest o tym fakcie informowany odpowiednim komunikatem.

Dodatkowo, oprócz komunikatów, kolor wpisywanych danych zmienia się z czarnego na czerwony. Poprawnie wpisane dane zmieniają kolor na zielony. Wypełnienie wszystkich pól powoduje wyświetlenie przycisku umożliwiającego wysłanie formularza.

Kod HTML tworzący formularz kontaktowy odpowiednio formatujemy w pliku CSS. Znaczniki zawierają komunikaty o błędach. Są one początkowo ukryte. Parametr **autofocus** wprowadzony w HTML5 służy do ustawienia kursora na polu tekstowym po wczytaniu strony umożliwiając natychmiastowe wprowadzanie danych.

```
<form id="contact_form">
  <ul>
     <li>
        <label for="i1">
     Imię:
        </label>
        <span class="error">
           <span class="10">Imię za krótkie</span>
           <span class="11">Proszę wpisać imię poprawnie</span>
        </span>
        <p>
           <input id="i1" type="text" name="imie" maxlength="15" autofocus>
        </p>
     </li>
     <li>
        <label for="i2">
     Email:
  </label>
        <span class="error">
```

```
            <span class="2">Niepoprawny adres  e-mail</span>
         </span>
         <p>
            <input id="i2" type="email" name="email">
  </p>
     </li>
     <li>
        <label for="message">
     Wiadomość:
  </label>
         <span class="error wiad">
            <span class="3">Wiadomość za krótka</span>
         </span>
  <p>
     <textarea name="tekst" id="message"></textarea>
  </p>
     </li>
     <span id="wymagane">Wszystkie pola są wymagane</span>
     <li class="slij">
        <button class="wyslij_kontakt button">Wyślij</button>
     </li>
  </ul>
</form>
```

Tworzymy tablicę **kontakt**, w której będą zapisywane dane z formularza oraz zmienne błędu danych, które będą przyjmować wartości 0 lub 1.

```
var kontakt = new Array();
var error_imie;
var error_email;
var error_wiadomosc;
```

Skrypt sprawdzający tekst wiadomość jest bardzo prosty . Jego zasada działania sprowadza się do sprawdzenia długości wpisanego tekstu. Skrypt weryfikujący imię oprócz sprawdzania długości kontroluje czy wpisywane są tylko litery.

Omówmy teraz kod skryptu sprawdzającego poprawność wpisanego adresu poczty elektronicznej. Najpierw sprawdzane jest czy użytkownik wprowadził jakiekolwiek dane (2). Jeśli nie zmienna **error_email = 1** (4) uniemożliwiając wysłanie formularza. Gdy adres e-mail został wpisany jest on porównywany z wzorcem[7] (7). Niepoprawnie wpisany adres zmienia kolor na czerwony (18), zostaje wyświetlona informacja o błędzie (17). Zmienna **error_email** w tym przypadku równa się jeden (19). Poprawnie wpisany e-mail zostaje zapisany w tablicy **kontakt** pod nazwą „Email” (12) i zmienia kolor na zielony (11). Zmienna **error_email** przyjmuje wartość zero (13). Wiadomość o błędzie zostaje ukryta (10).

```
1  $('#i2').keyup( function(){
2    if ($(this).val() == 0)
3    {
4     error_email = 1;
5     $('.2').hide();
6    }
7    else if ( new RegExp('^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0-9-]+
8     (\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$','i').test($(this).val())==true)
9    {
10    $('.2').hide();
11    $(this).css({'color':'green'});
12    kontakt['Email'] = $(this).val();
13    error_email = 0;
```

[7] Wzór **RegExp** zaczerpnięty z http://www.mkyong.com/regular-expressions/how-to-validate-email-address-with-regular-expression/, 20.04.2013.

```
14   }
15   else
16   {
17    $('.2').show();
18    $(this).css({'color':'red'});
19    error_email = 1;
20   }
21 });
```

Aby można było wysłać formularz wszystkie dane muszą być poprawne. Sprawdzany jest warunek czy wszystkie zmienne błędu danych są równe zeru. Jeśli wszystkie dane zostały wpisane poprawnie, wyświetli się przycisk umożliwiający ich wysłanie. Po kliknięciu użytkownik zostanie poinformowany o wysłaniu.

```
$('#contact_form').keyup(function(){
  if ( error_imie == 0 && error_email == 0 && error_wiadomosc == 0 )
    {
     $('.wyslij_kontakt').css({'visibility':'visible'});
     $('#wymagane').stop(true,true).fadeOut();
    }
  else
    {
     $('.wyslij_kontakt').css({'visibility':'hidden'});
     $('#wymagane').stop(true,true).fadeIn();
    }
});
```

4. Wniosek

Problemem podczas projektowania nowoczesnych witryn internetowych wykorzystujących najnowsze technologie są przeglądarki internetowe. Standardy takie jak HTML5 i CSS3 nie są jeszcze w pełni wspierane przez twórców topowych przeglądarek internetowych. Wymusza to niekiedy wprowadzanie do stron dodatkowych linijek kodu (zazwyczaj w plikach CSS) optymalizujących je kod kątem konkretnej przeglądarki. Wydłuża to czas projektowania strony i zmniejsza przejrzystość kodu.

Literatura

[1] Castledine E., Sharkie C.: *jQuery. Od nowicjusza do wojownika ninja,* Wyd. Helion, Gliwice 2012.

[2] Freeman E., Robson E.: *HTML5. Rusz głową!*, Wyd. Helion, Gliwice 2012.

[3] Hogan Brian P.: *HTML5 i CSS3. Standardy przyszłości*, Wyd. Helion, Gliwice 2011.

[4] Majkowski W.: *jQuery. Tworzenie animowanych witryn internetowych*, Wyd. Helion, Gliwice 2013

[5] Mikołajewski P.: *jQuery. Kod doskonały*, Wyd. Helion, Gliwice 2012.

[6] Wrotek W.: *CSS3. Kaskadowe arkusze stylów. Ćwiczenia praktyczne*, Wyd. Helion, Gliwice 2013.

[7] http://api.jquery.com/, *(dostęp 14.04.2013).*

[8] http://doman.art.pl/kursjs/, *(dostęp 12.04.2013).*

[9] http://msdn.microsoft.com/pl-pl/library/struktura-dokumentu-i-nowe-elementy-html5.aspx, dostęp 20.01.2013

[10] http://pl.wikipedia.org/wiki/HTML, *(dostęp 12.01.2013).*

[11] http://www.mblog.boo.pl/artykul-119-jquery-roznica-miedzy-documentready-a-windowload.html, *(dostęp 06.04.2013).*

[12] http://www.w3schools.com/, *(dostęp 26.01.2013).*