Instrukcja 5

1. Cel ćwiczenia

Obecnie coraz rzadziej dokumenty HTML posiadają uzupełnioną treść poprzez ich ręczną edycję. Celem niniejszego materiału będzie wprowadzenie rozdział treści strony od jej szablonu projektowego.

2. Teoria

Obecnym trendem jest podział poszczególnych elementów strony WWW na niezależne moduły – szkielet dokumentu tworzony za pomocą znaczników, stylizowanie dokumentu poprzez użycie pliku stylów kaskadowych (CSS), włączenie obsługi zdarzeń na stronie (pliki skryptów JavaScript) oraz dołączenie treści właściwej strony WWW. W niniejszym materiale zajmiemy się właśnie tym aspektem (poprzednie zostały zrealizowane poprzez poprzednie materiały).

Odłączenie tekstu strony od ciała dokumentu, a nawet takich elementów jak zdjęcia tworzące galerię czy np. przyciski odnoszące użytkownika do kolejnych podstron, to dziś standard. Dzięki temu zabiegowi programista nie musi po każdorazowym dodaniu nowej treści (podstrony) aktualizować ręcznie całego menu bądź też, przy potrzebie zmiany tekstu, edytować całego dokumentu HTML gdzie ingerencja może być przyczyną uszkodzenia kodu szkieletu (np. błędne usunięcie jednego ze znaczników).

Coraz częściej treści strony (oraz elementy dodatkowe, jak przyciski czy pozycje menu) ładuje się dynamicznie poprzez element XMLHttpRequest(). Pozwala on na pobranie fragmentów bądź całych treści wskazanych plików z serwera strony WWW i załadowanie ich poprzez metody DOM w odpowiednie miejsce.

Niniejszy materiał wskaże możliwości ładowania i umieszczania treści strony w technologii NoSQL, czyli z wykorzystaniem plików JSON i/lub XML.

INFORMACJA: Sama technologia NoSQL to bardzo szerokie zagadnienie. Materiał ten ma jedynie zasygnalizować czytelnikowi istnienie takiej technologii oraz wskazać jej najprostszą formę. Obecnie na rynku oprogramowania można spotkać pełne rozwiązania bazujące na tej metodzie gromadzenia i przetwarzania danych informatycznych.

Największym atutem rozwiązań NoSQL jest szybkość dostępu do interesujących nas danych – w przeciwieństwie do SQL nie obowiązują tutaj zasady atomiczności, spójności czy też integralności. Pliki z danymi formatujemy w sposób wygodny dla naszych potrzeb, nie zaś dla serwera baz danych. Pliki z danymi mogą być rozdzielane i łączone zarówno w obrębie pojedynczego serwera jak i maszyn w sieci lokalnej czy poprzez sieć rozległą. To kolejna zaleta baz tego typu – nie mają problemu ze skalowalnością tzw. poziomą (czyli rozproszoną), pozwalającą bez problemu tworzyć całe klastry z danymi. W przypadku skalowalności pionowej wszystko zależy od wykorzystywanego rozwiązania; samodzielnie projektowane działania zależą od obsługującego ich języka po stronie serwera (oraz samego serwera, takiego jak np. Apache czy ngix), z kolei przy pełnych rozwiązaniach (jak OrientDB, Oracle czy MongoDB) wszystko zależne jest od implementacji samego rozwiązania (jedno/wielowątkowe, sposób podziału plików, mechanizmy blokowania zasobów itd.)

Do wad NoSQL można zaliczyć:

- powtarzalność danych – ponieważ dane w plikach należą do grupy tzw. płaskich, to dane z poszczególnych pól (jak imię czy nazwisko) będą każdorazowo dla kolejnych wierszy informacji powielane. To z kolei powoduje tworzenie nadmiarowych danych (i zajmowanie przez nie dodatkowej przestrzeni dyskowej)
- brak mechanizmów zabezpieczania danych – w większości przypadków sami musimy zadbać o kopie zapasowe danych (chyba, że używamy gotowych rozwiązań)

- brak mechanizmów blokady danych – sami musimy utworzyć mechanizm blokowania zapisu i odczytu danych, co jest szczególnie ważne w większych rozwiązaniach z wieloma użytkownikami mającymi dostęp do zapisu danych (sam odczyt nie wymaga mechanizmów zabezpieczających)
- wydajność przeszukiwania – sami musimy zadbać o indeksowanie informacji (co coraz częściej ma też miejsce w baza SQL)

Na koniec należy dodać, że bazy tego typu są coraz częściej i coraz chętniej wybierane do zastosowań tzw. Big Data czyli zbierania i przetwarzania wielkiej ilości danych celem zdobywania dodatkowej wiedzy np. dla sztucznej inteligencji (analiza rozmów internetowych, zachowań ludzi w sklepach itp.)

2.1 Pliki XML

Jednym z dość powszechnie stosowanych standardów przechowywania informacji są pliki XML. Sam XML jest rozszerzalnym językiem znaczników (eXtensible Markup Language). Język ten wyrósł z innego, starszego języka znaczników – SGML (Standard Generalized Markup Language).

SGML zakłada, że wszystkie znaczniki są udokumentowane i posiadają swoją definicję oraz funkcje w tworzonym pliku (dokumencie). Chociaż istnieje wiele narzędzi i implementacji tego języka (np. dokumenty HTML, doc, odt) to niemal żadne z nich nie zawiera pełnego zestawu znaczników i atrybutów przewidzianych w standardzie.

Mnogość znaczników, atrybutów i właściwości oraz trudność w implementacji doprowadziły do powstania opisywanego XML. Pierwotnie zakładany był jako rozszerzenie języka SGML; obecnie rozpatrywany jest jako niezależny język. Najlepszym przykładem niezależności jest brak obustronnej kompatybilności – o o ile parsery XML mogą czytać dokumenty SGML (muszą zostać jednak poprawnie sformułowane), o tyle pliki XML nie będą w przytłaczającej większości mogły być czytane przez parsery SGML (brak informacji o użytych znacznikach).

Przykład pliku XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<wiadomosci>
	<wiadomosc>
		<imie>Jan</imie>
		<nazwisko>Długosz</nazwisko>
		<adres>jd@serwer.pl</adres>
		<do>testowy@nazwa.org</do>
		<temat>Wiadomość testowa</temat>
		<tresc>Tutaj może być bardzo długa, jak i krótka wiadomość tekstowa. Znaczniki nie ograniczają przestrzeni danych, zawartych pomiędzy nimi bajtów czy znaków specjalnych (powinny być bez problemu przechowane). Dodatkowo istnieje możliwość przechowywania danych binarnych (np. załączniki) jednak w celu zachowania zgodności z plikami tekstowymi najlepiej jest je zakodować do postaci BASE64. </tresc>
	</wiadomosc>
	<wiadomosc>
		<imie>Joanna</imie>
		<nazwisko>Waluś</nazwisko>
		<adres>moj-adres@onet.pl</adres>
		<dot>adaml@op.pl</do>
		<imiedo>Adam</imiedo>
		<nazwiskodo>Lont</nazwiskodo>
		<temat>Wiadomość testowa</temat>
```

<tresc>Tutaj może być bardzo długa, jak i krótka wiadomość tekstowa. Znaczniki nie ograniczają przestrzeni danych, zawartych pomiędzy nimi bajtów czy znaków specjalnych (powinny być bez problemu przechowane). Dodatkowo istnieje możliwość przechowywania danych binarnych (np. załączniki) jednak w celu zachowania zgodności z plikami tekstowymi najlepiej jest je zakodować do postaci BASE64. </tresc>
</wiadomosc>
<wiadomosc>
<imie>Jan</imie>
<nazwisko>Długosz</nazwisko>
<adres>jd@serwer.pl</adres>
<do>testowy@nazwa.org</do>
<temat>Wiadomość testowa</temat>
<tresc>Tutaj może być bardzo długa, jak i krótka wiadomość tekstowa. Znaczniki nie ograniczają przestrzeni danych, zawartych pomiędzy nimi bajtów czy znaków specjalnych (powinny być bez problemu przechowane). Dodatkowo istnieje możliwość przechowywania danych binarnych (np. załączniki) jednak w celu zachowania zgodności z plikami tekstowymi najlepiej jest je zakodować do postaci BASE64. </tresc>
</wiadomosc>
</wiadomosci>

Jak widać na powyższym przykładzie nazwy znaczników możemy definiować sami. Łatwo zauważyć iż dokument zawsze posiada jeden element główny (w tym wypadku jest nim znacznik wiadomosci). Następnie kolejne elementy są w nim zawarte (w przykładzie odpowiada za nie znacznik wiadomosc). Znaczniki wiadomosc możemy tutaj rozpatrywać jako wiersz danych (bądź jako linię danych - w zależności od systemu zapisu). Kolumnami danych są wszystkie znaczniki zawarte w naszym wierszu danych. Jak łatwo zauważyć druga porcja danych różni się od pierwszej i drugiej dodatkowymi kolumnami danych. W notacji XML nie jest to błędem – traktowane jest to na równi z wartością NULL w bazach SQL.

Powyższy przykład nie pokazuje pełnych możliwości XML. Obecnie pliki te mogą wykorzystywać atrybuty (tak samo jak wszystkie języki z rodziny SGML),przestrzenie nazw, słowniki znaczników czy przestrzenie z danymi binarnymi (cdata). Istnieje nawet możliwości kolorowania składni pików XML w ten sam sposób, jaki ma miejsce w plikach HTML (przy pomocy języka XSLT - eXtensible Stylesheet Language Transformations). Jednak dla przechowywania danych dla strony WWW możliwości te nie mają większego znaczenia – wszelkie transformacje wykonywać będziemy przez mechanizmy przeglądarki (HTML/JS/CSS).

2.2 Pliki JSON

Kolejnym popularnym typem przechowywania danych są obiekty JSON (JavaScript Object Notation). Pliki te popularność swą czerpią z bezpośredniego dopasowania do drzewa DOM – załadowanie danych w tej notacji powoduje znaczne szybsze wczytywanie ich i/lub operacje na nich niż działania na danych XML/HTML/tekstowych, które najpierw muszą zostać „znalezione”, następnie „uchwycone” (np. do zmiennej bądź zmiennej niejawnej) by JS mogło z nich skorzystać.

Jako zaletę JSON wymienia się mniejszy plik bazy. Oczywiście jest to prawdziwe twierdzenie. Zapis następuje do właściwości obiektu, które nie muszą być powielane na końcu wartości (zakończeniem danych jest cudzysłów i/lub przecinek). Ponadto JSON pozwala na używanie tablic, które nie są obecne w XML (ani nawet w PL/SQL).

Pliki JSON mogą przechowywać różne typy wartości danych. W zasadzie jedynym ograniczeniem jest brak wsparcia dla przechowania funkcji, wartości nieokreślonych (undefined) oraz zmiennych datowych (aczkolwiek można je zapisywać jako ciągi znakowe). Trzeba pamiętać, że każdy typ danych można zapisać w JSON jako ciąg znakowy – JS bez problemu poradzi sobie z

rzutowaniem typów (wtedy możemy przechowywać zarówno funkcje, jak i wartości nieokreślone; problemem będzie natomiast wykorzystanie tego typu danych na stronie WWW).

Przykład pliku JSON:


```
[
	{"photos":
		[
			{
				"photo":".\/zdjecie1.jpg",
				"mini":".\/zdjecie1_mini.png",
				"alt":"Jaki\u015b przyk\u0142adowy tekst zdj\u0119cia"
			},
			{
				"photo":".\/zdjecie2.jpg",
				"mini":".\/zdjecie2_mini.png",
				"alt":""
			},
			{
				"photo":".\/galleries\/zdjecie1.JPG",
				"mini":".\/galleries\/zdjecie1_mini.png",
				"alt":"Sample text"
			}
		]
	}
]
```


Powyższy przykład należy czytać następująco:
tablica jednoelementowa (nawias kwadratowy to tablica, klamra to definicja obiektu) zawierająca jeden element o nazwie (bądź wedle notacji obiektowej właściwości) photos. Element ten zawiera tablicę, w której zdefiniowane są 3 obiekty, każdy posiadający właściwości photo, mini oraz alt.

Jak więc można wywnioskować każdy zapisany obiekt JSON musi zostać zamknięty w tablicy. Niekiedy można spotkać się z zapisem nietablicowym (bez rozpoczęcia od nawiasów kwadratowych) jednak może on powodować problemy z odczytem zarówno po stronie skryptów JavaScript jak i innych języków (np. PHP).

Same obiekty JSON zamykane są w nawiasach klamrowych. Pojedynczy nawias może zawierać TYLKO jeden obiekt. Sam obiekt natomiast może zawierać właściwość, która jako wartość będzie posiadać kolejną tablicę obiektów bądź pojedynczy obiekt.

Bardzo ważne jest by po ostatnim elemencie nie stawiać przecinka – może to spowodować nieoczekiwane zachowanie ze strony parsera (będzie oczekiwał kolejnego elementu).

Inny przykład zapisu pliku JSON:


```
[
	[
		{
			"desc":"Nazwa oddziału firmy 1",
			"tels":"Telefon kom\u00f3rkowy: (+48) 777-000-111;Fax: (+44) 777-987-
900",
			"addressname":"",
			"comment":"",
			"address":"Pozytywna 12;;;;;;",
```

```
                "town":"Spała",
                "postalcode":"44-444",
                "postoffice":"Spała",
                "workHours":"Pn-Pt: 12-17;Sob: 10-12;",
                "lat":"0",
                "lon":"0",
                "zoom":"0",
                "deleted":"0"
        }
    ],
    [
        {
                "desc":"Przykładowa nazwa oddziału 2",
                "tels":"Telefon kom\u00f3rkowy: (+12) 999-888-222",
                "addressname":"Tutaj podaj adres oddziału do wyświetlenia",
                "comment":"Komentarz dla wprowadzajacego",
                "address":"Przykładowa nazwa oddziału 2;Życzna;12;45-678;Warszawa;;",
                "town":"Warszawa",
                "postalcode":"45-678",
                "postoffice":"Warszawa",
                "workHours":"",
                "lat":"19.675",
                "lon":"21.976",
                "zoom":"17",
                "deleted":"0"
        }
    ]
]
```


W tym wypadku mamy tablicę główną zawierającą dwa elementy tablicowe (niepełna tablica dwuwymiarowa – główny element zawsze jest tylko jeden). Każdy element tablicy zawiera jeden obiekt JSON, które posiadają proste właściwości (pojedyncze wartości tekstowe). Należy zaznaczyć, że niektóre z nich mogłyby mieć inny typ – np. lat, lon czy zoom można by zapisywać jak typ liczbowy.

Podsumowując, pliki JSON z poworu mogą wydawać się nieco bardziej chaotyczne od układu XML, jednak możliwość łączenia typów, dodawanie tablic, obiektów podrzędnych (mogących zawierać w sobie dodatkowe obiekty), lepsza integracja z JavaScript (jest to integralna część tego języka) oraz mniejsza objętość (zamiast długich nazw obiektów klamry i przecinki) sprawiają, że ten typ przechowywania danych jest wybierany zdecydowanie częściej (przynajmniej w przypadku stron i aplikacji HTML5).

2.2.1 Odczyt danych JSON przez JavaScript

W celu odczytania plików JSON można wykorzystać element obiektu jQuery. Poniższy kod wczytuje wskazany plik i tworzy z niego tablicę JSON. Warto zauważyć, że wynik pobrania JSON z pliku o nazwie plikjson.js (rozszerzenie może być dowolne gdyż pliki z obiektem traktowane są jako zwykłe tekstowe) zapisywany jest do zmiennej result. W tym momencie zmienna ta przechowuje pełny obiekt JSON, z którego możemy korzystać. Jeżeli Obiekt jest prosty (zawiera proste elementy typu właściwość → wartość możemy, tak jak w przykładzie, wykorzystać funkcję each (dla każdego), gdzie obiekt result rozbijamy na klucz (obiekt JSON może być traktowany jako

tablica) oraz wartość klucza (odpowiednio wartości i i field jako parametry anonimowej funkcji). Kolejne wartości z tablicy zostaną wypisane do obiektu z ID #obiekt.

```
$(document).ready(function(){
      $.getJSON("plikjson.js", function(result){
         $.each(result, function(i, field){
            $("#obiekt").append(field + " ");
         });
      });
});
```

Przykład ten można modyfikować na własne potrzeby (np. w przypadku gdy posiadamy dodatkowe tablice jako wartości we właściwościach bądź obiekty zależne).

2.3 Zastosowanie techniki AJAX na stronie WWW.

Innym sposobem na wczytanie zawartości plików JSON jest wykorzystanie bezpośrednio modułu JavaScript – XMLHttpRequest. Rozwiązanie to może być znacznie elastyczniejsze niż poprzez jQuery,a na pewno szybsze.

Realizacji odczytu można dokonać na kilka sposobów. Każdy z nich zostanie przedstawiony i krótko omówiony.

1) Poprzez funkcję eval:

```
function getJSON(fileAddress) {
        var contentFile;
        if (window.XMLHttpRequest)
                contentFile = new XMLHttpRequest();
        else
                contentFile = new ActiveXObject("Microsoft.XMLHTTP");
        contentFile.onreadystatechange = function () {
                if(contentFile.readyState === 4) {

                        if(contentFile.status === 200 || contentFile.status === 0) {
                                var json = eval(contentFile.responseText)[0];
                                var content = document.createElement('div');
                                for (var i = 0; i < json['photos'].length; i++) {
                                        var fig = document.createElement('figure');
                                        var img = document.createElement('img');
                                        img.src = json['photos'][i]['mini'];
                                        img.alt = json['photos'][i]['alt'];
                                        fig.appendChild(img);
                                        if (json['photos'][i]['alt'] !== '') {
                                                var figcaption =
document.createElement('figcaption');
                                                figcaption.className = 'imgcaption';
                                                figcaption.innerHTML = json['photos'][i]['alt'];
                                                fig.appendChild(figcaption);
                                        }
                                        content.appendChild(fig);
                                }
                                document.body.appendChild(content);
```

```
                        return;
                    }
                    else {
                        console.log('Odczyt się nie powiódł');
                        return;
                    }
                }

        }
        contentFile.open("POST", site, true);
        contentFile.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        contentFile.send(post);
}
```

Powyższa funkcja pozwala na pobieranie dowolnej treści plików z naszego/odległego serwera WWW. W wykorzystanym kontekście służy jedynie do obsługi jednego rodzaju plików JSON – do tworzenia galerii na stronie WWW. Fragment przetworzenia obiektu do postaci DOM został pogrubiony. Pozostałe linie są uniwersalne i mogą współpracować z dowolnym innym pobieranym plikiem.

W tym przypadku dynamicznie dotwarzamy nowe obiekty na stronie WWW (div, figure, img oraz figcaption). Utworzone elementy uzupełniamy pobranymi danymi i wklejamy je na stronę. Tutaj warto zaznaczyć, że funkcja eval działa we wszystkich przeglądarkach i z każdym parserem JS. NIESTETY JEST RÓWNIEŻ DOŚĆ NIEBEZPIECZNA – potrafi bowiem parsować złośliwy kod JS, fałszywe dane itp. To z kolei może dać podstawę do ataku na tworzoną przez z nas stronę (przykładowo wyświetlą się łudząco podobne fałszywe dane, które pozwolą wykraść dane użytkownika – jak login czy hasło). Dlatego zaleca się nie wykorzystywać tej funkcji w swoich projektach (chyba, że dokładnie wiemy co robimy i jakie mogą być tego następstwa).

2) Zmodyfikowany natywny kod JS (przedstawiony fragment to część kodu poprzednio pogrubiona):

```
var json = (new Function('return ' + contentFile.responseText ))()[0];
var content = document.createElement('div');
for (var i = 0; i < json['photos'].length; i++) {
        var fig = document.createElement('figure');
        var img = document.createElement('img');
        img.src = json['photos'][i]['mini'];
        img.alt = json['photos'][i]['alt'];
        fig.appendChild(img);
        if (json['photos'][i]['alt'] !== '') {
                var figcaption = document.createElement('figcaption');
                figcaption.className = 'imgcaption';
                figcaption.innerHTML = json['photos'][i]['alt'];
                fig.appendChild(figcaption);
        }
        content.appendChild(fig);
}
document.body.appendChild(content);
```

Proszę zauważyć, że różnicy uległa tylko jedna linijka kodu! Nowe podejście, w przeciwieństwie do funkcji eval jest bardzo bezpieczne. Polecenie new Function powoduje bowiem utworzenie nowego obieku z danych zwracanych przez parametr. Słowo return wzięte jest w apostrof gdyż

parametr ma czysto tekstową postać. Na końcu funkcji można zaobserwować dodane okrągłe nawiasy. Bez nich stworzylibyśmy jedynie definicję obiektu tworzącego bez wywołania go (nawiasy powodują bezpośrednie wykonanie konstruktora). Na końcu mamy charakterystyczne odwołanie do zerowego elementu tablicy (czyli pobranie elementu JSON). Reszta kodu działa tak samo jak w poprzednim przykładzie. Proszę pamiętać, że starsze implementacje JS mogą nie zaakceptować tego rozwiązania. Proszę także pamiętać, że to rozwiązanie zostało dodane do parsowania danych w jQuery (wersje poniżej 3.x.x używają eval)

3) Użycie obiektu JSON:

```
var json = JSON.parse(contentFile .responseText)[0];
var content = document.createElement('div');
for (var i = 0; i < json['photos'].length; i++) {
	var fig = document.createElement('figure');
	var img = document.createElement('img');
	img.src = json['photos'][i]['mini'];
	img.alt = json['photos'][i]['alt'];
	fig.appendChild(img);
	if (json['photos'][i]['alt'] !== '') {
		var figcaption = document.createElement('figcaption');
		figcaption.className = 'imgcaption';
		figcaption.innerHTML = json['photos'][i]['alt'];
		fig.appendChild(figcaption);
	}
	content.appendChild(fig);
}
document.body.appendChild(content);
```

W ostatnim przykładzie wykorzystany został nowy obiekt JSON. Jest to najbezpieczniejsza forma Niestety niektóre starsze przeglądarki mogą mieć problem z obsługą kodu – wymaga on bowiem dodatkowej biblioteki, która nie zawsze dołączana jest do parsera.

Należy mieć na uwadze, że przykład 2 i 3 można ze sobą połączyć poprzez złączenie LUB. W takim wypadku pierwsze należy podać rozwiązanie 3, a jeżeli nie dojdzie ono do skutku (brak biblioteki) wykorzystane zostanie podejście numer 2.

Jak widać rozwiązań na dostęp do plików JSON jest całkiem sporo. Należy pamiętać, że przedstawione powyżej rozwiązania będą znacznie szybsze w działaniu niż te proponowane z jQuery. Z kolei jQuery będzie działało mniej awaryjnie na większej ilości przeglądarek i parserów JS.

3. Zadania do realizacji.

W tworzonym projekcie należy treść strony odseparować od aktualnego kodu HTML. W tym celu najlepiej będzie utworzyć plik json, w którym znajdywać się będą kolejne treści naszych podstron. W drugiej kolejności należy utworzyć plik json odpowiadający za aktualny stan naszej galerii zdjęć oraz menu głównego. Układy plików json jest dowolny, ważne jednak by minimalnie zawierały:
a) dla treści stron – treść strony (wraz z jej wewnętrznym kodem HTML, jeżeli konieczny), datę dodania, datę ostatniej modyfikacji
b) galeria – odnośnik do zdjęcia, odnośnik do miniatury, podpis obrazka
c) menu głowne – nazwa przycisku, adres odniesienia

Proszę przy tym pamiętać o odpowiedniej obsłudze większej ilości tworzonych obiektów z pliku json – przykładowo wskutek dodania kolejnych odnośników w menu głównym bądź zdjęć w galerii.

UWAGA! Istnieje spore prawdopodobieństwo, że większość przeglądarek nie będzie chciało wczytywać lokalnych plików poprzez technikę AJAX. Dlatego należy korzystać z przeglądarki Mozilla Firefox (pozwala na ładowanie lokalnej zawartości poprzez AJAX) bądź wrzucić stronę na serwer WWW (np. zainstalowany lokalnie – XAMPP bądź WAMP, do których odnośniki podane są poniżej w materiałach).

MATERIAŁY:
https://msdn.microsoft.com/pl-pl/dn912483.aspx
http://www.cs.put.poznan.pl/szfrancuzik/data/uploads/nosql.pdf
http://www.computerworld.pl/news/W-ktora-strone-skalowac,383163.html
http://dembol.org/blog/projektowanie/katalog-wzorcow-architektonicznych/wzorce-perspektywy-alokacji/skalowanie/
https://www.w3.org/TR/xml-stylesheet/
https://www.w3schools.com/xml/default.asp
https://www.w3schools.com/js/js_json_intro.asp
https://www.apachefriends.org/pl/index.html
http://www.wampserver.com/en/