Macie już swój genialny pomysł na stronę lub całą aplikację internetową. Świetnie. Będzie oszałamiać użytkownika. Będzie świetna wizualnie i łatwa w obsłudze. AJAX jest pierwszym skojarzeniem - słusznie. To wygodna, elastyczna, efektywna i efektowna technologia. Ale trzeba ją wprowadzić rozważnie. Trzeba się zastosować do podejścia zwanego postępującym ulepszaniem. Polska Wikipedia mówi na ten temat niewiele - angielska, pod hasłem progressive enchacement - już więcej. O co więc chodzi.

Jeśli macie jakieś lepsze tłumaczenie niż progresywne ulepszanie (z polskiej Wikipedii) lub postępujące ulepszanie, zaproponowane przeze mnie, dajcie znać w komentarzach. Na pewno da się wymyślić jakąś lepszą nazwę na to, o co nam chodzi.

Teraz nastąpi chwila przydługiego wstępu. Uwaga: jeśli trafiliście tu, bo chcecie się dowiedzieć czegoś o technicznej realizacji podejścia postępującego ulepszania, pomińcie te akapity. Jeśli nie wiecie, o co chodzi i po co to wszystko, poświęćcie trochę czasu, żeby je przeczytać.

Najważniejsze zasady przyświecające podejściu postępującego ulepszania to:

1. Podstawowa treść i funkcjonalność powinna być dostępna we wszystkich przeglądarkach.
2. Cała treść powinna być dostępna w lekkim kodzie znaczników.
3. Dodatkowe opcje wyglądu powinny być zawarte w zewnętrznych, dołączanych plikach stylów CSS.
4. Dodatkowe funkcjonalności powinny być zawarte w zewnętrznych, dołączanych plikach JavaScript.

Co do pierwszego punktu - jeśli rzeczywiście chcecie zawładnąć światem za pomocą swojej aplikacji, nie możecie ograniczyć się do jednej przeglądarki - to chyba oczywiste. Co prawda właśnie z horyzontu znika nielubiane przez nas, webdeveloperów, zjawisko pod tytułem IE6. Znika, bo jeśli Google nie będzie jej obsługiwać, nie ma szans przetrwania. Ale nadal pozostaje

kwestia palety aktualnych przeglądarek. Dodatkowo, jeśli celujemy z naszym oprogramowaniem w rynek urządzeń przenośnych - pomyślmy o dostępności dla przeglądarek na te urządzenia.

Co do udostępniania funkcjonalności, wydaje się, że z wykorzystanie bibliotek typu jQuery można przestać się przejmować różnicami w obsłudze skryptów przez rożne przeglądarki. A co, jeśli obsługi skryptów zabraknie?

Tu wielu zrobiło duże oczy... Po co ktoś miałby wyłączać obsługę skrypty. Pomijam fakt, że zdarzają się ludzie wojujący z JavaScriptem. Są jednak osoby korzystające z urządzeń wspomagających przy obsłudze stron internetowych - mogą nie uwzględniać wielu naszych wspaniałych funkcjonalności napisanych w JS - w ten sposób nasza aplikacja przestaje mieć dla nich sens.

Do wielu pięknych i młodych przyszłych milionerów zarabiających na swojej aplikacji to tłumaczenie może nie docierać. Może zatem dotrze to - awarie mają mniejszą siłę rażenia, jeśli strona zawiera alternatywne rozwiązania dla pełnej funkcjonalności.

Drugi punkt - lekki kod. Treść w czystej postaci powinna być zawarta w kodzie znaczników - do tego został stworzony. Nie jest tu aż tak istotne, czy będzie to HTML, XHTML czy może XML przekształcany za pomocą XSLT. Powody - po pierwsze: indeksowanie. Choć mechanizmy Google nie są powszechnie znane, to niemal na pewno, jeśli bot dostanie się w łatwy sposób do treści - można liczyć na wyższe rankingi. Z pokrewnego powodu nie robimy layoutów na tabelkach, ale to temat na inny artykuł. Po drugie: znów dostępność - tym razem chwilę zatrzymajmy się przy urządzeniach przenośnych - pakiety internetowe telefonii komórkowej wciąż nie są tak tanie, jak byśmy tego chcieli. Z tego powodu wielu użytkowników wyłącza ściąganie niepotrzebnych danych - jak obrazki, skrypty etc. W walce o klienta, każdy KB danych, których nie trzeba ściągać, aby zapoznać się z treścią witryny, może być istotny.

Punkty 3 i 4 to rozwinięcie tematu i wyjaśnienie - co zrobić, gdy chcemy mieć jakieś bajery.

OK, trochę teorii i marudzenia za nami, teraz pora rzeczywiście coś zaprogramować w duchu postępującego ulepszenia.

Mam co prawda mnóstwo pomysłów na genialne aplikacje, ale żadnym z nich się nie podzielę.

Stworzymy za to prostą witrynę prezentującą dane na dwóch stronach, z menu w postaci zakładek.

Kod HTML:

{codecitation} <li class="selected"> treść 1 treść 2 Treść dostępna pod pierwszą zakładką {/codecitation}

w drugiej stronie dwie zmiany: klasa selected trafi do drugiego elementu listy, a treść kontenera content będzie inna.

{codecitation}   treść 1 <li class="selected"> treść 2 Treść dostępna pod drugą zakładką {/codecitation}

Zapisujemy to jako pliki index.html i tresc2.html - nazwy są oczywiście nieistotne, o ile będziemy pamiętać o nich przy odnośnikach.

Wyświetlenie tej witryny w obecnej postaci nie stanowi żadnego problemu - mamy dostępną pełną funkcjonalność. Dodatkowo strona jest lekka - nie chodzi o rozmiar mierzony w KB, ale o narzut struktury na treść - rzeczywiście póki co treść jest w mniejszości, ale jedno zdanie chyba lepiej napisać na twitterze niż tworzyć własną stronę. Gdy przybędzie treści, okaże się, że wspomniany narzut jest niewielki. Bot wyszukiwarki na pewno ucieszyłby się (gdyby potrafił) widząc taką strukturę.

Tu uwaga: HTML ma sporo znaczników, które mają swój sens - tu widzicie tego przykład. Niech elementy list służą do wyświetlania list. Paragrafy - do paragrafów, a kontenery div - do tworzenia podziału strony. Nie opierajmy wszystkiego tylko na div-ach, paragrafach bądź listach - niech się uzupełniają. Nawet, uwaga uwaga, tabelki nie są złe, jeśli używa się je do właściwych celów - prezentacji danych tabelarycznych.

OK, nasza strona działa. Jestem pewien, że żadna współczesna przeglądarka (a i starsze również) nie będzie miała problemów z jej wyświetleniem. Ale przecież nie żyjemy w epoce hateemela łupanego, tylko w web 2.0. Gdzie wrażenia użytkownika? Przecież jeśli zobaczy taką stronę, natychmiast ją zamknie. (to prawda, choć z wyjątkiem wspomnianych użytkowników urządzeń przenośnych - oni mogą być nam wdzięczni).

Chcielibyśmy, żeby odnośniki pojawiały się w postaci kart zakładek, aktywna zakładka niech ma inny kolor, a kontener content był otoczony obwódką. Miło byłoby też, żeby całość była wyśrodkowana na stronie. W kodzie HTML przemyciłem już odpowiednie oznakowanie klas i identyfikatorów, pora je ostylować:

{codecitation class='brush: css'}* { margin: 0; padding: 0; } div#tabs { margin: auto; width: 70%; } ul { float: left; list-style-type: none; } li { float: left; padding: 10px;

border: 1px solid black; } li.selected { background-color: grey; } div#content { clear: both; border: 1px solid black; padding: 10px; }{/codecitation}

Do HTMLa w sekcji HEAD dodajemy

{codecitation}<link href="style.css" rel="stylesheet" type="text/css" />{/codecitation}

Strona wyświetla się tak, jak chcieliśmy. Lista przybrała postać zakładek, aktywna zakładka jest szara. Przy odrobinie wyobraźni może to udawać zakładki. Jeśli jednak nie chcemy liczyć na odrobinę wyobraźni, popracujmy nad stylami, żeby wszystko to wyglądało jakoś sensowniej. Dla potrzeb tego przykładu, musi wystarczyć to, co jest.

Czas na bajery. Nie oczekujcie oczywiście jakichś niesłychanych rzeczy - co można zrobić z dwoma zdaniami na dwóch stronach. Ale AJAX będzie i to, w perspektywie, całkiem praktyczny.

Nie chcemy przeładowywać całej strony. Chcemy, żeby ładowanie nowej treści odbyło się w sposób niewidoczny dla użytkownika. Stracimy co prawda funkcjonalność przycisku wstecz, ale to element często poświęcany w imię AJAXowego interfejsu.
Podłączamy bibliotekę jQuery. Można to zrobić na mnóstwo sposobów - o tym poczytacie gdzie indziej (np. na youthcoders.net lub jquery.com). W tym przykładzie, prosto z katalogu strony, więc w HEAD dodajemy

{codecitation}<script src="jquery.js" type="text/javascript"></script>{/codecitation}

Oprogramowanie zakładek

{codecitation}$(document).ready(function() { $('.link').click(function(event) { var href = $(this).attr("href"); var currentLink = $(this).parent().parent().children('.selected'); var thisLink = $(this).parent(); $.ajax({ url: href, dataType: "html", success: function(response) { currentLink.removeClass("selected"); thisLink.addClass("selected"); $('#content').html($('#content',response).html()); } }); event.preventDefault(); }) });{/codecitation}

Po kolei:

Linia 3. - definiujemy reakcję na kliknięcie na łączu w zakładce

Linia 5. - pobieramy adres odnośnika w zakładce

Linia 7. - aktualnie podświetlona zakładka - będziemy chcieli ją wygasić

Linia 9. - kliknięta zakładka - będziemy chcieli ją podświetlić

Teraz (linia 11.) wykorzystamy obiekt AJAX udostępniany przez jQuery. To co widzimy, to wywołanie jego konstuktora i przekazanie mu parametrów. Najważniejszy jest parametr success - to funkcjia anonimowa, która zostanie wywołana, jeśli przeglądarka odbierze wynik zapytania wysłanego AJAXem. Ten wynik możemy obsłużyć - to właśnie parametr response tej funkcji. Nazwa jest całkiem dowolna - sami ja ustalamy, ale musimy się jej potem trzymać.

Najpierw obsługujemy zakładki - gasimy jedną i zapalamy drugą.

Następnie, w linii 23., rzecz trochę karkołomna. $('#conent').html(treść) spodowduje, że do kontenera content wstawiona zostanie treść - ale nie możemy przekazać tam całej otrzymanej strony - przecież zawiera ona też zakładki i całą resztę. Interesuje nas tylko zawartość takiego samego kontenera z tamtej strony. Wydobądźmy go więc. Muszę przyznać, że byłem w szoku, jak proste jest to zadanie, gdy użyjemy jQuery (jQuery często wywołuje u mnie szok, przyznaję). Funkcja $('#conent').html() - czyli gdy nie ma parametru - zwraca zawartość, zamiast ją ustawiać. A wskazanie, że oczekujemy zawartości z otrzymanej strony, a nie z pliku, w którym się aktualnie znajdujemy - wystarczy po przecinku za selektorem podać zmienną otrzymaną w odpowiedzi na żądanie. Ta prostota jest zaskakująca.

Ale to jeszcze nie koniec. Jeśli w tym miejscu przerwalibyśmy, wszytsko nie miałoby sensu. Kliknięcie łącza najpierw pobrałoby stronę AJAXem, a potem i tak przeładowało ją. Funkcja event.preventDefault() wyłącza typowe działanie odnośnika (uwaga na parametr event funkcji anonimowej w linii: XYZ).

To cała istota postępującego rozszerzenia - mieliśmy łącze, które spełniało swoje zadanie.

Odbieramy mu je za pomocą JavaScriptu i zastępujemy lepszą funkcjonalnością. Jest to pewne i bezpieczne, bo skoro JavaScript zdołał odbebrać łączu funkcjonalność, to mamy pewność, że jest włączony i obsługiwany - może więc tę funkcjonalność zastąpić. Podobnie jest z ukrywaniem treści - jeśli chcesz, żeby jakiś kontener rozwijał się po kliknięciu - ukryj go skryptem, a nie stylem. Wtedy, jeśli uda się go ukryć, to na pewno uda się go też rozwinąć. A gdyby skrypty miały nie zadziałać, treść będzie odsłonięta.

Zapiszmy powyższy skrypt w pliku script.js i podłączmy do naszych plików HTML.

{codecitation}<script src="script.js" type="text/javascript"></script>{/codecitation}

Aplikacja (ok, to trochę za duże słowo - raczej strona) jest gotowa. Z jednej strony mamy fajne wykorzystanie AJAXa - można je rozszerzać na wiele zakładek i stron, z drugiej zapewniliśmy dostępność, a zawartość merytoryczna jest dostępna dla każdego, niezależnie od tego, jak i gdzie ogląda naszą stronę.

A teraz przyznajmy, że to co zrobiliśmy nie ma sensu. Nadal mamy 2 strony, które trzeba utrzymywać (choć maksymalnie to uprościliśmy, trzymając wspólne części - skrypt i style - w osobnych plikach), nadal przez sieć podróżuje cała strona, a nie tylko potrzebny nam jej fragment. Prawdziwe cuda można zdaiałać przy pomocy np. skryptów PHP - żądanie AJAX może modyfikować url dodając jakiś parametr, który strona serwera rozpozna i w odpowiedzi wystawi tylko potrzebne dane, być może dynamiczne, aktualne itp. Jednak ten przykład ze statycznymi stronami pokazuje metodę i podejście - postępujące ulepszenie.

Podsumowanie:

- Zrób swoją strone prosto - mało kodu, dużo sensu, pełna funkcjonalność dostępna bez skryptów, stylu etc.
- Dodaj style i skrypty w zewnętrznym pliku - łatwiej tym zarządać, w perpektywie łatwiej zmieniać, a użytkownicy, którzy tego nie chcą, nie będą musieli ściągać dodatkowych plików.
- Niech dopiero dodane skrypty wyłączą funkcjonalność - bo wtedy mamy pewność, że potrafią ją zastąpić.

I jeszcze miejsce, gdzie można poczytać więcej o temacie, po angielsku:

http://en.wikipedia.org/wiki/Progressive_enhancement