Poprawiony artykuł dostepny na blogu autora " MVC w JavaScript zbudowane na jQuery ".

Kod HTML coraz częściej naszpikowany jest wieloma artybutami pomocnymi przy wdrażaniu kodu JS do projektu. Czy nie da się z tym czegoś zrobić? Dodatkowo bardzo często kod JS jest "brudny" przez mieszanie wartw: danych, prezentacji i logiki. W małym projekcie to znośne, ale w dużych zaczyna być nie do ogarnięcia. Spróbuję pokazać co można zmienić w tej kwestii.

Strona "spaghetti"

Dawniej stosowało się określenia "spaghetti HTML", mając na myśli kod HTML naszpikowany artybutami nadającym stronie wygląd. Aby temu zapobiegać zaczęto stosować CSS. W przypadku JS także można zostawić kod HTML czystym od wszelkim artybutów "zdarzeniowych" (onclick, onmouseover itp.). Najpierw kod, gdzie tego nie zrobię i artybuty będą zaszyte w kodzie HTML:

{codecitation class='brush: html'}

<html> <head> <meta http-equiv="content-type" content="text/html; charset=utf-8"/> </head> <body> Tu się coś zmieni <button onclick="document.getElementById('content').innerHTML='Cześć'">Akcja 1</button> </body> </html>

{/codecitation}

Po kliknięciu na przycisk w kontenerze #conent zostanie podmieniony ciąg znaków. Czy jednak nie wydaje Ci się to nieelegancke?

Zebranie wszystkich akcji w jednym miejscu

Spróbujmy do tego podejść trochę inaczej. Zamiast wstrzykiwać kod JS do artybutów znaczników, dajmy ten kod w jednym miejscu strony. W poniższym przykładzie użyję bibliteki jQuery. Jeśli nie używałeś jej nigdy, zachęcam do krótkiego tutorialu .

{codecitation class='brush: html'}

<html> <head> <meta http-equiv="content-type" content="text/html; charset=utf-8"/> <script type="text/javascript" src="http://code.jquery.com/jquery-1.4.2.min.js"></script> <script type="text/javascript"> $(document).ready(function() { $('#action').click(function() { $('#content').html('Cześć'); }); }); </script> </head> <body> Tu się coś zmieni <button id="action">Akcja 1</button> </body> </html>

{/codecitation}

Kod HTML od razu stał się bardziej przejrzysty. Jednak nikt nie powiedział, że ten cały system będzie tak prosty. Mało który skrypt JS nie korzysta dzisiaj z ajaksa, często zmienia duże kawałki kodu HTML operując na modelu DOM. Może warto więc byłoby zrobić to trochę inaczej:

{codecitation class='brush: html'}

<html> <head> <meta http-equiv="content-type" content="text/html; charset=utf-8"/> <script type="text/javascript" src="http://code.jquery.com/jquery-1.4.2.min.js"></script> <script type="text/javascript"> $(document).ready(function() { var oController = yController(); $('#action').click(oController.clickHandler); }); var yController = function() { function fClickHandler() { $('#content').html('Cześć'); } return { clickHandler : fClickHandler }; }; </script> </head> <body> Tu się coś zmieni <button id="action">Akcja

1</button> </body> </html>

{/codecitation}

Tym sposobem ograniczyliśmy do jednej linii kod odpowiedzialny za przypisanie zdarzeniu jakiejś funkcji. Jednak skrypt rozrósł się nam trochę. Na dobrą sprawę w obiekcie yController dzieje się to samo, co działo się wcześniej w ciele anonimowej funkcji. Spróbujmy dodać jeszcze dodatkowe obiekty - jeden, który będzie "gmerał" w modelu DOM, a drugi który będzie uzyskiwał wymagane dane.

Model - View - Controller

{codecitation class='brush: html'}

<html> <head> <meta http-equiv="content-type" content="text/html; charset=utf-8"/> <script type="text/javascript" src="http://code.jquery.com/jquery-1.4.2.min.js"></script> <script type="text/javascript" src="mvc.js"></script> </head> <body> Tu się coś zmieni <button id="action1">Akcja 1</button> </body> </html>

{/codecitation}

Oraz kod pliku mvc.js

{codecitation class='brush: js'}

$(document).ready(function() { var oController = yController(); $('#action1').click(oController.clickHandler); }); var ySrc = function() { function fGetLabel(fSuccess, fErorr) { // tu jakis ajax, ja podaje na sztywno dane, // tak jakbym przekazal do funkcji callback var fakeData = { result : "cześć" }; fSuccess(fakeData); } return { getLabel : fGetLabel }; }; var yView = function() { function fDisplayContent(data) { $('#content').html(data.result); } return { displayContent : fDisplayContent }; }; var yController = function() { var oSrc = ySrc(), oView = yView(); function fClickHandler() { var label = oSrc.getLabel(oView.displayContent); } return { clickHandler : fClickHandler }; };

{/codecitation}

Jak widać bardzo często korzystam tu z faktu że w JS bardzo przyjemnie przekazuje się funkcje jako parametry do innych funkcji. Zobacz linię 37. Kiedy stwierdzisz, że co dane powinny się pojawić gdzie indziej ograniczysz liczbę zmian do jednego miejsca - w konstruktorze obiektu yView. A być może - patrząc przyszłościowo - także kod z `displayContent' da się wykorzystać jeszcze w kilku miejscach?

Przykład wykorzystania tego podejścia w większym projekcie:

- http://wdp-client.googlecode.com/svn/trunk/inc/js/frontController.js
- http://wdp-client.googlecode.com/svn/trunk/inc/js/view.js
- http://wdp-client.googlecode.com/svn/trunk/inc/js/dataSrc.js