Niedawno opublikowałem mój walidator formularzy yFormValidator . W tym artykule pokażę dokładniej możliwości tego bardzo prostego frameworka walidacyjnego.

Jeśli nie interesuje Cię część teoretyczna, skocz do praktyki .

Predefiniowane klasy

yFormValidator posiada kilka z góry zdefiniowanych klas służących do walidacji. Są to:

- 'not empty' - niepusty ciąg znaków
- 'alphanumeric' - litery, cyfry oraz podkreślenie, bez białych znaków
- 'integer' - liczby całkowite (dodatnie i ujemne)
- 'decimal' - liczby rzeczywiste (dodatnie i ujemne)
- 'date' - data w formacie dd/mm/rrrr, od roku 1600 [uwzględnia lata przestępne, liczby dni w miesiącach itp.]
- 'email' - adres e-mail

Nie gwarantuję, że stworzone przeze mnie wyrażenia regularne są prawidłowe i na pewno nie przepuszczą żadnych nieprawidłowych danych.

Własne wyrażenia regularne

Prócz wyżej wymienionych zdefiniowanych klas można podać także własne wyrażenie regularne, istnieją 3 sposoby:

1. rule : /regexp/flags (literał wyrażeń regularnych)
2. rule : new RegExp("regexp", "flags") (konstruktor obiektu RegExp)
3. rule : "regexp" (string)

Kiedy walidacja?

Pole formularza jest walidowane gdy:

1. w obiekcie `items' występuje klucz taki sam jak wartość atrybutu `name' elementu formularza
2. w konfiguracji ustawiono `required = true'
3. pole ma co prawda `required = false' ale posiada jakąś wartość

Prosty przykład

{codecitation class='brush: html'}<html> <head> <title>Walidacja formularza RegExp</title> <meta http-equiv="content-type" content="text/html;charset=utf-8" /> <script type="text/javascript" src="yFormValidator.js"></script> <script type="text/javascript"> // funkcje potrzebne do odpowiedniego kolorowania formularza function invalid() { this.style.borderColor = "red"; return false; } function valid() { this.style.borderColor = "green"; return true; } window.onload = function() { var data = { form : 'info-form', error : function(rule) { return invalid.call(this); }, success : function(rule) { return valid.call(this); }, items : { 'name' : { rule : 'not empty', required : true }, 'age' : { rule : 'integer', required : false }, 'height' : { rule : 'decimal', required : false }, 'date' : { rule : 'date', required : true }, 'email' : { rule : 'email', required : true }, 'site' : { rule : (/^http:///), required : false } } }; yFormValidator(data); } </script> </head> <body> <form action="#" id="info-form"> Imię i nazwisko*: <input type="text" name="name" /> (dowolny niepusty ciąg znaków)
 Wiek: <input type="text" name="age" /> (liczba całkowita)
 Wzrost (w metrach): <input type="text" name="height" /> (liczba rzeczywista)
 Data dd/mm/yyyy*: <input type="text" name="date" /> (dd/mm/yyyy), powyżej 1600
 E-mail*: <input type="text" name="email" /> (poprawny składniowo adres e-mail)
 WWWW : <input type="text" name="site" /> (adres zaczynający się od http)
 <input type="submit" value="gotowe" /> </form> </body> </html>{/codecitation}

Zobacz demo online

Uruchamiając ten przykład możemy dostrzec kilka problemów:

- wiek, czy wzrost mogą być ujemne, a mimo to przejdą walidację.
- data może być poprawna, np. "10.10.2000", a mimo to nie przejdzie (wymaganym znakiem jest "/" - nie ".")
- wzrost może być poprawny, np. "1,65", a mimo to nie przejdzie walidacji (wymagana jest ".", a nie ","). Dodatkowo warto byłoby mieć tylko dwa miejsca po przecinku, a "1.789997788" nadal jest poprawną liczbą rzeczywistą

(dodatnią)

Aby to naprawić, spróbujmy wykorzystać funkcje obsługi zdarzeń `error' oraz `success' dla wskazanych pól.

Obsługa zdarzeń

Najpierw naprawię "poprawnie" podany wiek. W funkcji obsługi `success' operator `this' wskazuje na element formularza o atrybucie `name="age"'. Możemy zatem łatwo sprawić jego wartość:

{codecitation class='brush: javascript'}... 'age' : { rule : 'integer', required : false, success : function() { if (this.value < 0) { // mogłoby być: // this.style.borderColor = 'red'; // return false return invalid.call(this); } return valid.call(this); } }, ...{/codecitation}

Zobacz demo online

W tym wypadku w przypadku, kiedy walidacja przejdzie pomyślnie jeszcze raz upewniamy się, czy oby na pewno wszystko jest poprawnie. Jeśli ktoś podał ujemną liczbę zwracamy `false', co jest rozumiane przez skrypt tak samo jakby pole nie przeszło walidacji wyrażeniem regularnym.

Ciekawić Cię może dziwne wywołanie funkcji `valid' oraz `invalid'. Dzięki wykorzystaniu metody `call' wewnątrz tych funkcji operator `this' będzie wskazywał na element fomrularza (czyli na to samo, co wewnątrz funkcji `error'/`success').

- https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Function/Call

{codecitation class='brush: javascript'}... 'date' : { rule : 'date', required : true, error : function(rule) { var val = this.value.replace(/[-. \]/g, "/"); if (rule.test(val)) { this.value = val; return valid.call(this); } return invalid.call(this); } }, ...{/codecitation}

Zobacz demo online

Został jeszcze jeden przypadek "prawidłowych - nieprawidłowych danych". Kiedy ktoś poda nieszczęsny przecinek zamiast kroki. Prawdziwa zmora programistyczna. A przecież można łatwo załatać:

{codecitation class='brush: javascript'}... 'height' : { rule : 'decimal', required : true, success : function(rule) { if (this.value > 0) { this.value = Number(this.value).toFixed(2); return valid.call(this); } return invalid.call(this); }, error : function(rule) { var val = this.value.replace(/[,]/g, "."); if (rule.test(val) && val > 0) { this.value = Number(val).toFixed(2); return valid.call(this); } return invalid.call(this); } }, ...{/codecitation}

Zobacz demo online

Warto przypomnieć, że sam framework ma rozmiar 3,29kB, wersja skompresowana jedynie **1,64kB**
.

Przydatne linki

- Pobierz wszystkie dema
- Zobacz yFormValidator na svn
- inny wpis o yFormValidator
- oryginlany artykuł: http://www.yarpo.pl/2011/06/06/wykorzystanie-yformvalidator-walidacja-formularzy/