bootcamp online

JAVASCRIPT

Witaj w trzecim tygodniu Bootcampu

Czas rozpocząć trzeci tydzień zmagań w Bootcampie JavaScript. Mam nadzieję, że poprzedni tydzień był dla Ciebie ciekawy oraz zawierał wiele cennych informacji. W tym tygodniu poznasz technologię **Ajax**, a także powiązane z nią tematy takie jak **JSONP** czy **ciasteczka**. Dodatkowo, zagłębimy się w temat **wyrażeń regularnych**, czyli niezwykle potężnej funkcjonalności niemal każdego języka programowania. Nie zabraknie również dobrych praktyk pracy z kodem JavaScript.

Piotr Palarz

Piotr Palarz

--- Zadania na tydzień 3 ---

*Zadania domowe spakuj ZIPem i umieść na kanale **#prace_domowe** na Slacku.*
W przypadku zadań, gdzie niezbędny jest dodatkowy kod HTML i CSS, nie musisz się skupiać na tym, aby strona wyglądała pięknie. Jeśli masz czas i chęci, dopracuj swój projekt, ale najważniejszy pozostaje i tak kod JavaScript.

1. Polyfill metody repeat dla String

W standardzie **EcmaScript 2015** (o którym więcej w 5 i 6 tygodniu Bootcampu) pojawiła się nowa metoda dostępna na obiektach typu `String` o nazwie `repeat`. Jej użycie wygląda następująco:

```
“hej ”.repeat(3) // zwraca “hej hej hej ”
```

Metoda ta jest dostępna we wszystkich nowoczesnych przeglądarkach internetowych, ale aby poćwiczyć rozszerzanie wbudowanych typów, utwórz jej **polyfill**. W kodzie sprawdź najpierw czy taka metoda już w przeglądarce została zaimplementowana, a jeśli nie, to dopisz własną funkcję, która będzie mogła być na dowolnym stringu wywołana w podany wyżej sposób.

Podpowiedź: rozszerzaj prototyp konstruktora `String`. Przy sprawdzeniu czy taka metoda już istnieje, w nowoczesnych przeglądarkach otrzymasz odpowiedź pozytywną. Jeśli zatem napiszesz odpowiedni warunek, to nie będzie można przetestować Twojej metody. Z tego powodu, zamiast `repeat` możesz ją nazwać `repeatt`.

2. Wrapper Toggler dla elementów z drzewa DOM

Do przygotowanego pod adresem http://pastebin.com/hUK5tnh3 kodu dodaj konstruktor (klasę) o nazwie `Toggler`. Przy tworzeniu nowych jej instancji z użyciem słowa kluczowego `new` (jak możesz zobaczyć w przygotowanym kodzie) przekazywać będziemy selektor. Za jego pomocą należy znaleźć na stronie odpowiedni element (skorzystaj z metody `document.querySelector`) i zapisać go w nowo stworzonym obiekcie. Następnie dodaj 3 metody. Pierwsza z nich o nazwie `getElem` powinna po prostu zwrócić znaleziony wcześniej element. Metoda `show` i `hide` powinny kolejno **pokazywać** i **ukrywać** element.

Jeśli wszystko wykonasz poprawnie, kod który został już napisany powinien działać bez żadnych modyfikacji. Zauważ, że do elementu o identyfikatorze **button** zostało przypisane zdarzenie kliknięcia. Taki element musisz wstawić na stronę, podobnie jak i element, którego selektor zostanie przekazany przy tworzeniu nowego obiektu klasy `Toggler`.

3. Ajaxowy polyfill dla funkcji fetch

Napisz **polyfill** dla funkcji fetch (nie będziemy się tutaj trzymać dokładnie tego, w jaki sposób ona działa, stworzysz jedynie prostą jej wersję). Wykorzystaj obiekt `XMLHttpRequest` w ten sposób, aby docelowo korzystanie z funkcji `fetch` wyglądało następująco:

```
fetch("url", function(data) {

    console.log("Sukces");

    console.log(data);

}, function(err) {

    console.log("Wystąpił błąd!");

    console.log(err);

});
```

a zatem jako pierwszy argument przekazujemy adres **URL** (wyślij pod niego zapytanie **GET**), jako drugi funkcję, którą należy wykonać **jeśli wszystko się powiedzie** (przekaż jej pobrane dane), a jako trzeci funkcję, która wykona się **na wypadek błędu** (przekaż jej obiekt z błędem lub komunikat tekstowy). W nowoczesnych przeglądarkach istnieje już funkcja `fetch`, a zatem aby jej nie nadpisywać, możesz nadać jej inną nazwę, np. `fecz`. Jako adres URL, z którego pobierane bedą dane, możesz wykorzystać http://code.eduweb.pl/bootcamp/users/

4. Funkcja getJSON

Mając już rozwiązanie zadania trzeciego, przepisz swój kod tak, aby stworzyć nową funkcję o nazwie `getJSON`. Idea działania i użycie będą bardzo podobne:

```
getJSON("url", function(data) {

    console.log("Sukces");

    console.log(data);

    // typeof data powinno zwrócić “object”

}, function(err) {

    console.log("Wystąpił błąd!");

    console.log(err);

});
```

Różnica jest taka, że tym razem pod parametrem `data` przekazanym w funkcji callback, powinien się znajdować javascriptowy obiekt, a nie string przysłany z serwera. Serwer, który przygotowaliśmy pod tym adresem: http://code.eduweb.pl/bootcamp/users/ działa tak, że po otrzymaniu standardowego zapytania **GET**, np. kiedy wpiszesz ten adres w przeglądarce lub wyślesz zapytanie Ajaxem, zwróci dane zawarte w kodzie HTML. Jeśli jednak przy wysyłaniu zapytania dodasz nagłówek `Accept: application/json`, to serwer zwróci te same dane, ale w formacie **JSON** (bez kodu HTML). Z poziomu Twojego kodu JavaScript, otrzymane Ajaxem dane to cały czas typ `String`, ale za pomocą metody `JSON.parse` możesz ten string łatwo zamienić na javascriptowy obiekt. To jest właśnie Twoje zadanie. Wysyłając żądanie do serwera, dodaj wspomniany wcześniej nagłówek za pomocą metody setRequestHeader, a otrzymane dane sparsuj za pomocą `JSON.parse` i dopiero wtedy przekaż jako parametr `data` do funkcji callback.

5. Parser adresu URL

Stwórz funkcję o nazwie `getPage`, która pozwoli odczytać numer strony z adresu URL. Dane na temat adresu URL strony, na której wykonuje się Twój kod JavaScript, możesz odczytać za pomocą obiektu `window.location`. Dopisz do adresu Twojej strony **?page=2** i wykorzystując odpowiednią właściwość obiektu `location`, a także odpowiednie wyrażenie regularne, zwróć z funkcji `2` (typ `Number`) lub `null`, jeśli query string **?page=X** nie będzie podany lub będzie niepoprawny, np. **?page=tekst**

Użycie funkcji powinno wyglądać następująco:

```
// dla adresu np. http://localhost/test/?page=2
getPage(); // zwraca 2

// dla adresu np. http://localhost/test/
getPage(); // zwraca null

// dla adresu np. http://localhost/test/?page=nieliczba
getPage(); // zwraca null
```

Ajax i JSON

Od “Ajaxowego boom’u”, który miał miejsce w 2005 roku wiele się zmieniło. Wcześniej strony internetowe, aby wyświetlić jakąś nową zawartość, potrzebowały całkowitego przeładowania. Dzisiaj w wielu przypadkach jest zupełnie inaczej. Dodajemy produkt do koszyka, a on pojawia się tam natychmiast. Wszystko to dzięki asynchronicznemu wywołaniu, które realizowane jest z poziomu kodu JavaScript. Użycie takiej techniki ma wiele zalet. Po pierwsze wszystko realizowane jest dużo szybciej. Użytkownik po dodaniu produktu do koszyka czy wpisaniu komentarza na portalu społecznościowym, widzi rezultat swojego działania natychmiast. Po drugie odciążamy serwer. W klasycznym modelu po dodaniu produktu do koszyka, przeglądarka wysłałaby zapytanie HTTP (np. metodą POST), serwer przyjąłby dane, dodał produkt do koszyka i wygenerował widok (kod HTML) całej strony od nowa. Oznacza to, że do przeglądarki z powrotem zostałaby wysłana cała strona, gdzie być może zmieniłaby się tylko jedna ikonka, pokazująca liczbę produktów dodanych do koszyka. Przyznasz, że to naprawdę nierozsądne marnowanie zasobów serwera i czasu potencjalnego klienta, gdy można taką operację wykonać dużo szybciej, prawda?

Skoro wiemy już, że Ajax pozwala przesyłać dane do i z serwera, dobrze byłoby przesyłać je w jakimś powszechnie stosowanym formacie. I tak też jest! Kiedyś popularny był format **XML**, ale już od wielu lat króluje **JSON**. To lekki format wymiany danych, który jest naturalny dla języka JavaScript, ale jest także dobrze “rozumiany” przez technologie serwerowe takie jak np. PHP. Dodawanie ajaxowych komponentów do stron internetowych to jednak nie wszystko. Dzięki rozwojowi licznych frameworków tworzonych w języku JavaScript, a także możliwościom Ajaxa, dzisiaj tworzone są bardzo zaawansowane aplikacje internetowe, które działają wprost w przeglądarce i coraz częściej mogą konkurować z natywnymi aplikacjami. Ich zaletą jest jednak fakt, że nie wymagają instalacji na urządzeniu użytkownika.

Ajax w praktyce

Wiemy już co nieco o samym Ajaxie, ale co to tak naprawdę jest? Jak z tego skorzystać? Pamiętasz, że język JavaScript to jedno, a API danego środowiska to drugie? Świetnie!

Ajax **nie jest** częścią języka JavaScript. Jest to jedno z wielu **API**, które udostępnia przeglądarka internetowa. Skoro zatem nasz kod jest wykonywany w jej środowisku, to podobnie jak w przypadku Obiektowego Modelu Dokumentu, mamy dostęp do specjalnych obiektów, które Ajaxa udostępniają.

Dokładnie jest to konstruktor (funkcja) o nazwie `XMLHttpRequest`. Jeśli wiesz już na czym polega programowanie obiektowe w języku JavaScript, to znasz pojęcie klasy (inaczej funkcji konstruującej) i instancji (obiektu utworzonego z tej klasy). `XMLHttpRequest` jest w tym przypadku klasą, która wywołana z użyciem słowa kluczowego `new`, zwróci nowy obiekt (instancję), który zawiera wszystkie potrzebne właściwości i metody, aby korzystać z Ajaxa.

Istotą Ajaxa jest asynchroniczność. Oznacza to, że możemy “poprosić” przeglądarkę internetową, by wysłała zapytanie HTTP o określonym przez nas typie, pod określony adres URL. Dodatkowo możemy przekazać jej dane, które chcemy pod ten adres wysłać. Na koniec podajemy również funkcję, którą chcemy wywołać, gdy cała operacja się zakończy. W tym miejscu słowo klucz to “gdy”. Wysyłanie zapytań do serwera i odbieranie z niego danych może być bardzo szybkie, ale także może trwać bardzo długo. Czasami po kilkunastu sekundach możemy doświadczyć tzw. timeout’u i zapytanie się nie powiedzie. Nie chcielibyśmy w tym momencie zablokować wykonywania pozostałego kodu JavaScript, prawda?

Wyobraź sobie, że do jakiegoś przycisku przypisaliśmy funkcję obsługującą zdarzenie kliknięcia. Być może w ten sposób pokazujemy jakąś ukrytą zawartość. Jeśli w między czasie wysłalibyśmy zapytanie Ajax, które nie byłoby asynchroniczne, to tak długo aż ono się nie zakończy, użytkownik naszej strony nie mógłby kliknąć we wspomniany przycisk, który miał spowodować wykonanie się jakiegoś kodu JavaScript.

Z asynchronicznością wiążą się tzw. funkcje zwrotne (ang. *callback functions*). Doświadczyłeś już ich użycia w przypadku obsługi zdarzeń. Kiedy przypisujemy zdarzenie kliknięcia do jakiegoś elementu, to również podajemy funkcję callback, która ma się

wykonać, gdy kliknięcie będzie miało miejsce. Nie wiemy kiedy to będzie i czy w ogóle się wydarzy. Podobnie jest z Ajaxem. Mówimy "*wyślij zapytanie pod **ten** adres, a **gdy** otrzymasz odpowiedź z serwera, wykonaj **tę** funkcję*".

Połączenie różnych API

Skoro wiesz już mniej więcej jak działa **Ajax**, i że pozwala nam wysyłać asynchroniczne zapytania do serwera, a także wiesz, że język JavaScript w połączeniu z **DOM API** pozwala nam np. tworzyć nowe elementy HTML, to wyobraź sobie taki scenariusz. Za pomocą Ajaxa pobieramy z serwera wyłącznie dane, bez kodu HTML jak ma to miejsce w klasycznym modelu. Kiedy mamy już dane, najlepiej w formacie JSON, możemy na ich podstawie zbudować widok HTML i wstrzyknąć go na stronę. W ten sposób serwer zajmuje się wyłącznie serwowaniem i odbieraniem danych, a klient ich wyświetlaniem. No i dokładnie w taki sposób buduje się dziś nowoczesne aplikacje webowe. Aby całe to zadanie było prostsze, niemal każdego dnia powstają nowe frameworki, które cały ten proces wspomagają. Warto jednak korzystając z nich, wiedzieć jak to wszystko działa "pod spodem", a zatem nauka natywnych API przeglądarek jest naprawdę ważna.

Wyrażenia regularne

Wyrażenia regularne (ang. *regular expressions*) to jedno z najpotężniejszych narzędzi niemal każdego języka programowania. Nie zabrakło ich również w języku JavaScript. W przeciwieństwie do Ajaxa, tym razem mamy do czynienia z natywną konstrukcją języka. Oznacza to, że wyrażenia regularne nie są częścią API przeglądarki internetowej lub innego środowiska. Możesz z nich korzystać wszędzie tam, gdzie działa język JavaScript.

Podobnie jak w przypadku innych konstrukcji języka, takich jak np. tablice, wyrażenia regularne też są obiektami. Można je utworzyć na kilka sposobów, ale w istocie później działają tak samo. Tak jak inne obiekty, udostępniają nam swoje właściwości i metody, za pomocą których pracujemy.

A czym w zasadzie są wyrażenia regularne? Są to pewnego rodzaju schematy, które definiujemy. Wyobraź sobie, że chcemy wymusić na użytkowniku, aby w danym polu formularza wpisał kod pocztowy w formacie `xx-xxx`, gdzie każdy `x` to jedna cyfra. Jak

możemy to zrobić? Właśnie z użyciem wyrażeń regularnych! Pozwalają nam one zdefiniować pewny schemat, który w tym przypadku wyglądałby następująco: `/\d\d-\d\d\d/`. I choć ten zapis na początku wygląda dziwnie, to wyobraź sobie, że każde wystąpienie `\d` oznacza, że ma w tym miejscu być jedna cyfra (0-9). Widzisz zatem dwa takie wystąpienia, a później pauzę, która oznacza dosłownie: "*w tym miejscu ma być pauza*". Następnie definiujemy ponownie 3 cyfry. Po utworzeniu takiego wyrażenia regularnego, możemy pobrać z pola formularza string, który wpisał użytkownik i "przepuścić" go przez odpowiednią metodę obiektu reprezentującego wyrażenie regularne. Taka metoda może zwrócić nam `true` lub `false`, w zależności od tego, czy dany ciąg znaków pasuje do wzorca.

Wyrażenia regularne oferują dużo więcej możliwości i są powszechnie stosowane nie tylko do walidacji, ale także do wyszukiwania określonych fraz w tekście, itd. Idea jest taka, że tworzymy wzorzec i konfrontujemy go z tekstem, a więc w języku JavaScript z typem `String`.

Nie zniechęcaj się!

Wyrażenia regularne to temat trudny do zrozumienia od razu, a także do zapamiętania. Ja sam często nie pamiętam wszystkiego i wracam do ściągi, a przy konkretnym zadaniu rzadko robię coś z głowy, mając pewność, że zadziała. Najczęściej wykorzystuję narzędzie RegExr, w którym testuję tworzone przez siebie wyrażenie.

Najważniejsze jest zatem, abyś zobaczył co oferują wyrażenia regularne, a także jak wykorzystać ich podstawy. W ten sposób, kiedy staniesz przed problemem, w którym mogłyby się przydać, będziesz mógł poeksperymentować i napisać odpowiednie wyrażenie.

Sporo nowej wiedzy w tym tygodniu, zatem do dzieła!