jQuery

wprowadzenie, instalacja, dokumentacja, wyszukiwanie elementów, obsługa eventów, modyfikacja i wstawianie elementów, obsługa AJAX, integracja z REST API

Plan zajęć

1. Wprowadzenie
2. Instalacja
3. Dokumentacja
4. Wyszukiwanie elementów
5. Modyfikacja elementów
6. Obsługa eventów
7. Wstawianie elementów
8. Obsługa AJAX
9. Integracja z REST API

Wprowadzenie

jQuery to **biblioteka** JavaScript ułatwiająca manipulowanie elementami DOM, dodawanie animacji oraz wykonywanie asynchronicznych zapytań HTTP.

Autorem biblioteki jest **John Resig**.

Wydana w roku 2006.

Instalacja

- ➔ Pobierz plik z kodem jQuery ze strony https://jquery.com/download, następnie podlinkuj go w *index.html*

  lub

- ➔ Podlinkuj plik z kodem jQuery ze strony https://cdnjs.com/libraries/jquery

  lub

- ➔ Zainstaluj jQuery przy pomocy menadżera pakietów (npm, Yarn)

Dokumentacja

Oficjalna dokumentacja jQuery:

https://api.jquery.com

Alternatywna dokumentacja jQuery:

http://jqapi.com

DOMContentLoaded a jQuery

Zanim przystąpimy do manipulacji elementami DOM musimy sprawdzić, czy nasz dokument został załadowany.

https://javascript.info/onload-ondomcontentloaded

DOMContentLoaded a jQuery / Przykład 1

```
$(function () {
    /* nasz kod */
});
```

lub

```
$(document).ready(function () {
    /* nasz kod */
});
```

Wyszukiwanie elementów

Wyszukiwanie elementów

Znając selektory CSS łatwiej jest wyszukiwać elementy DOM z jQuery.

Przykład:

```
#top, .content, input[type="submit"], itd.
```

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors

Wyszukiwanie elementów / Przykład 1

```
// Znajdź element o id top
$(‘#top’);

// Znajdź wszystkie elementy div
$(‘div’);

// Znajdź wszystkie elementy z klasą box
$(‘.box’);
```

Wyszukiwanie elementów

Istnieją metody umożliwiające efektywniejsze wyszukiwanie elementów:

- ➔ **find()** wyszukuje potomków bieżącego elementu
- ➔ **children()** wyszukuje dzieci bieżącego elementu
- ➔ **first()** wyszukuje pierwszy element spośród bieżących elementów
- ➔ **last()** wyszukuje ostatni element spośród bieżących elementów
- ➔ **eq(n)** wyszukuje n-ty element spośród bieżących elementów

Wyszukiwanie elementów / Przykład 2

```
// Znajdź wszystkie inputy w elemencie o id top
$('#top').find('input');

// Znajdź dzieci elementu z klasą menu
$('.menu').children();

// Znajdź trzeci element li z listy ul
$('ul').find('li').eq(2);
```

Wyszukiwanie elementów

- ➔ **next()** wyszukuje następny sąsiadujący element
- ➔ **prev()** wyszukuje poprzedni sąsiadujący element
- ➔ **parent()** wyszukuje najbliższy element rodzica
- ➔ **siblings()** wyszukuje elementy sąsiadujące

Wyszukiwanie elementów / Przykład 3

```
// Znajdź element następujący po elemencie header
$(‘header’).next();

// Znajdź element rodzica elementu z klasą close
$(‘.close’).parent();

// Znajdź elementy sąsiadujące z elementem span
$(‘span’).siblings();
```

Cache’owanie elementów

Cache’owanie to popularna technika programistyczna polegająca na zapisywaniu wyszukanych elementów w zmiennych.

- ➔ Jeżeli korzystasz z elementu więcej niż raz, zapisz go w zmiennej.
- ➔ Jeżeli w międzyczasie w DOM zajdą jakieś zmiany, np. pod wyszukany element zostanie dodany inny, będziesz musiał ponownie go wyszukać, aby uzyskać dostęp do nowo dodanego elementu.

Cache’owanie elementów / Przykład 3

```
// Zapisz referencję do wyszukanego elementu w zmiennej
var addressForm = $(‘.address’);

// Wyszukaj element select w formularzu z klasą address
addressForm.find(‘select’);
```

Wyszukiwanie elementów

W jakim celu wyszukujemy elementy DOM?

- ➔ animacja
- ➔ nadawanie atrybutów
- ➔ stylowanie
- ➔ kopiowanie
- ➔ usuwanie

Modyfikowanie elementów

Modyfikowanie elementów

- ➔ **show()** pokazuje bieżący element
- ➔ **hide()** ukrywa bieżący element
- ➔ **addClass()** dodaje klasę/-y do elementu
- ➔ **removeClass()** usuwa klasę/-y z element
- ➔ **toggleClass()** dodaje lub usuwa klasę/-y z elementu

Modyfikowanie elementów / Przykład 1

```
$(‘.side-nav’).show();

$(‘.side-nav’).hide();

$(‘ul’).find(‘li’).addClass(‘list-item’);

$(‘.post’).children().removeClass(‘content’);

$(‘#power-switch’).toggleClass(‘hidden’);

$(‘input[type=”submit”]’).addClass(‘foo’).removeClass(‘bar’);
```

Modyfikowanie elementów

- ➔ **css()** pozwala czytać i ustawiać style CSS na bieżącym elemencie

Modyfikowanie elementów

➔ **css()** pozwala czytać i ustawiać style CSS na bieżącym elemencie

Przykład:

```
<input type="text" class="user-name">

var userName = $('.user-name');

userName.css({ color: 'red' });

userName.css('color'); // => rgb(255, 0, 0)
```

Modyfikowanie elementów

➔ **attr()** pozwala czytać i ustawiać atrybuty na bieżącym elemencie

Modyfikowanie elementów

➔ **attr()** pozwala czytać i ustawiać atrybuty na bieżącym elemencie

Przykład:

```
<input type="text" id="top-secret">

var topSecret = $('#top-secret');

topSecret.attr('type', 'password');

topSecret.attr('type'); // => password
```

Modyfikowanie elementów

- ➔ **val()** pozwala czytać i ustawiać wartość na bieżącym elemencie

Modyfikowanie elementów

➔ **val()** pozwala czytać i ustawiać wartość na bieżącym elemencie

Przykład:

```
<select name="animals" id="animals">
    <option value="cat">Cat</option>
</select>

var animalsSelect = $('#animals');
animalsSelect.val(); // => cat
```

Modyfikowanie elementów

➔ **val()** pozwala czytać i ustawiać wartość na bieżącym elemencie

Jeżeli użytkownik zmieni w jakiś sposób bieżącą wartość elementu, należy ponownie pobrać jego wartość.

Modyfikowanie elementów

➔ **val()** pozwala czytać i ustawiać wartość na bieżącym elemencie

Jeżeli użytkownik zmieni w jakiś sposób bieżącą wartość elementu, należy ponownie pobrać jego wartość.

jQuery nie synchronizuje wartości automatycznie!

Modyfikowanie elementów

- ➔ **data()** ustawia lub pobiera aktualną wartość atrybutu data

Modyfikowanie elementów

➔ **data()** ustawia lub pobiera aktualną wartość atrybutu data

Przykład:

```
<div data-last-value="27" data-hidden="true"></div>


$('div').data('lastValue'); // => 27
$('div').data('hidden', false);
$('div').data('hidden'); // => false
```

Modyfikowanie elementów

➔ **data()** ustawia lub pobiera aktualną wartość atrybutu data

Zauważ, że wieloczłonowy atrybut, np. *last-value*, jest odczytywany jako *lastValue*, tj. *$('div').data('lastValue')*.

Obsługa eventów

Obsługa eventów

Odpowiedniki metod addEventListener() i removeEventListener() w jQuery to:

- ➔ **on()** pozwala dodawać eventy do bieżącego elementu
- ➔ **off()** pozwala usuwać eventy z bieżącego elementu

Obsługa eventów / Przykład 1

```
<button class=”alert-btn”>Click!</button>

function hello() {  alert(’Hello World!’)  }

var alertButton = $(’.alert-btn’);

alertButton.on(’click’, hello);
alertButton.off(’click’, hello);
```

Obsługa eventów

Jeżeli klik powoduje przeładowanie dokumentu użyj preventDefault().

Przykład:

```
<button id="logout-btn">Log out</button>

$('#logout-btn').on('click', function (event) {
    event.preventDefault();
    /* nasz kod */
});
```

Obsługa eventów

Najczęściej wykorzystywane eventy to:

- **Mysz**

  click dblclick mouseenter mouseleave
- **Klawiatura**

  keypress keydown keyup
- **Form**

  submit change focus blur
- **Document/Window**

  load unload resize scroll

Wstawianie elementów

Wstawianie elementów

- ➔ **text()** wstawia ciąg znaków jako tekst
- ➔ **html()** wstawia ciąg znaków jako HTML

Wstawianie elementów

- ➔ **text()** wstawia ciąg znaków jako tekst
- ➔ **html()** wstawia ciąg znaków jako HTML

Przykład:

```
<div class="text-area"></div>
<div class="text-area"></div>


$('.text-area').first().text('<strong>Hello!</strong>!');
$('.text-area').last().html('<strong>Hello!</strong>');
```

Tworzenie nowych elementów

Tworzenie elmentów HTML przy pomocy jQuery jest bardzo proste.

Pamiętaj, że podobnie jak w przypadku metody *createElement()* nasz nowo utworzony element nie zostanie automatycznie podpięty do dokumentu.

Tworzenie nowych elementów / Przykład 1

```
var newHeading = $('<h2>Title</h2>', { class: 'hidden' });

var newParagraph = $('<p id="introduction"></p>');

newParagraph.text('Hello World!');
newParagraph.css('color', 'blue');
newParagraph.addClass('text-center');
```

Wstawianie elementów

- ➔ **before()** wstawia element przed bieżącym elementem
- ➔ **after()** wstawia element za bieżącym elementem

Wstawianie elementów

- **before()** wstawia element przed bieżącym elementem
- **after()** wstawia element za bieżącym elementem

Przykład:

```
[before]
<p class="text-center"></p>
[after]

$('p').before('<strong>Before!</strong>');
$('p').after('<strong>After!</strong>');
```

Wstawianie elementów

- ➔ **prepend()** wstawia element przed zawartością bieżącego elementu
- ➔ **append()** wstawia element za zawartością bieżącego elementu

Wstawianie elementów / Przykład 2

```
<p class="text-center">
    [prepend]
    <span>Hello World!</span>
    [append]
</p>

$('p').prepend('<strong>Prepend!</strong>');
$('p').append('<strong>Append!</strong>');
```

Wstawianie elementów

- ➔ **remove()** usuwa bieżący element wraz z jego zawartością
- ➔ **empty()** usuwa wszystkie dzieci bieżącego elementu, w tym tekst

Obsługa eventów - ciąg dalszy

Eventy a wstawiane elementy

jQuery pozwala automatycznie przypisywać eventy nowo wstawionym elementom.

Wystarczy skorzystać z alternatywnej sygnatury metody **on()**:

```
on(event, selector, function)
```

Eventy a wstawiane elementy

jQuery pozwala automatycznie przypisywać eventy nowo wstawionym elementom.

Wystarczy skorzystać z alternatywnej sygnatury metody **on()**:

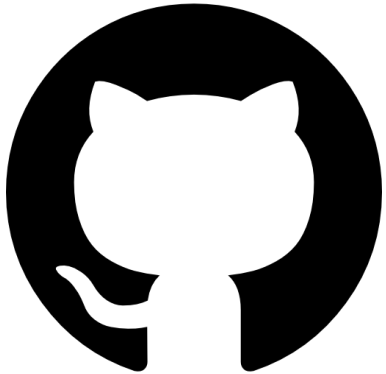
```
on(event, selector, function)
```

Metodę należy wywołać na rodzicu, który już istnieje!

Eventy a wstawiane elementy / Przykład 3

```
<section id="controls">
    <button class="ctrl-btn">Alert 1</span>
    <button class="ctrl-btn">Alert 2</span>
        [nowe elementy]
</section>

$('#controls').on('click', '.ctrl-btn', function () {
    alert('Boo!');
});
```



```
git clone https://github.com/kompedia/jquery-training.git
```

```
cd jquery-training
```

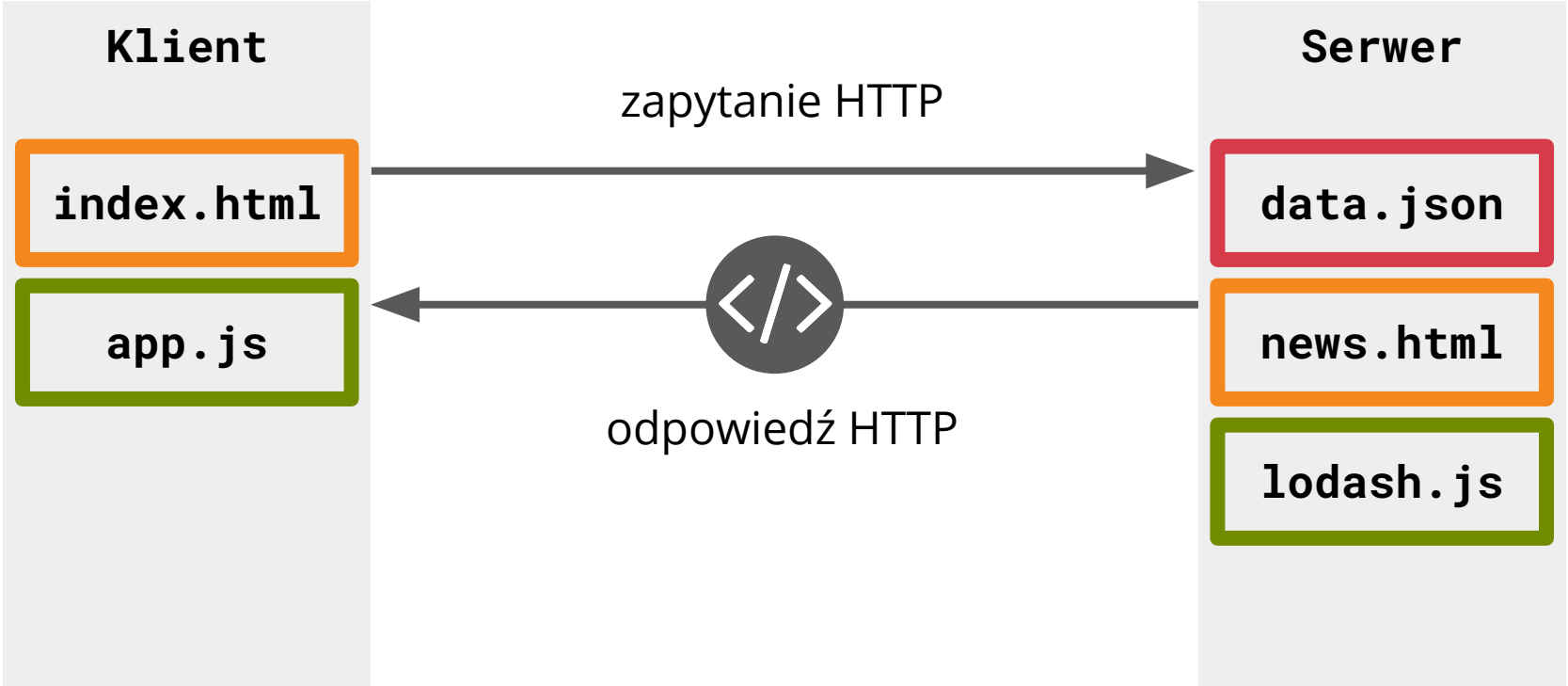
Obsługa AJAX

Obsługa AJAX

Asynchronous **J**avaScript **A**nd **X**ML

Technika wysyłania zapytań HTTP do serwera celem pobrania nowych treści bez konieczności przeładowania oglądanej strony internetowej.

Obsługa AJAX / Przykład 1



Obsługa AJAX

Najczęściej wykorzystywane metody HTTP to:

- ➔ **GET** pobieranie istniejącego zasobu
- ➔ **POST** tworzenie nowego zasobu
- ➔ **PUT** modyfikacja istniejącego zasobu
- ➔ **DELETE** usuwanie istniejącego zasobu

Obsługa AJAX

Najczęściej wykorzystywane metody HTTP to:

- ➔ **GET** pobieranie istniejącego zasobu
- ➔ **POST** tworzenie nowego zasobu
- ➔ **PUT** modyfikacja istniejącego zasobu
- ➔ **DELETE** usuwanie istniejącego zasobu

GET https://www.google.com/

Obsługa AJAX

➔ **ajax()** wszechstronna metoda służąca do wykonywania zapytań HTTP

Ta metoda jest bardzo rozbudowana i dlatego też warto szczegółowo zapoznać się z jej dokumentacją na *https://api.jquery.com*.

Obsługa AJAX / Przykład 2

```
var options = {
  url: 'www.example.com/foo-bar.html'
};

$.ajax(options)
  .done(function () { /* wykonaj jak wszystko pójdzie OK */ })
  .fail(function () { /* wykonaj jak coś pójdzie nie tak */ })
  .always(function () { /* wykonaj zawsze */ });
```

Obsługa AJAX / Przykład 3

```
var options = {
  url: 'www.example.com/users',
  method: 'POST',
  data: { name: 'John', lastname: 'Smith'}
};

$.ajax(options)
  .done(function () { /* wykonaj jak wszystko pójdzie OK */ })
  .fail(function () { /* wykonaj jak coś pójdzie nie tak */ });
```

XMLHttpRequest a jqHXR

Począwszy od jQuery 1.5, obiekt **jqXHR** stanowi pokrycie przeglądarkowego obiektu **XMLHttpRequest**.

http://api.jquery.com/jQuery.ajax/#jqXHR

Integracja z REST API

Dziękuję za uwagę.