



dr inż. Michał Malinowski
michal.malinowski@warszawa.merito.pl

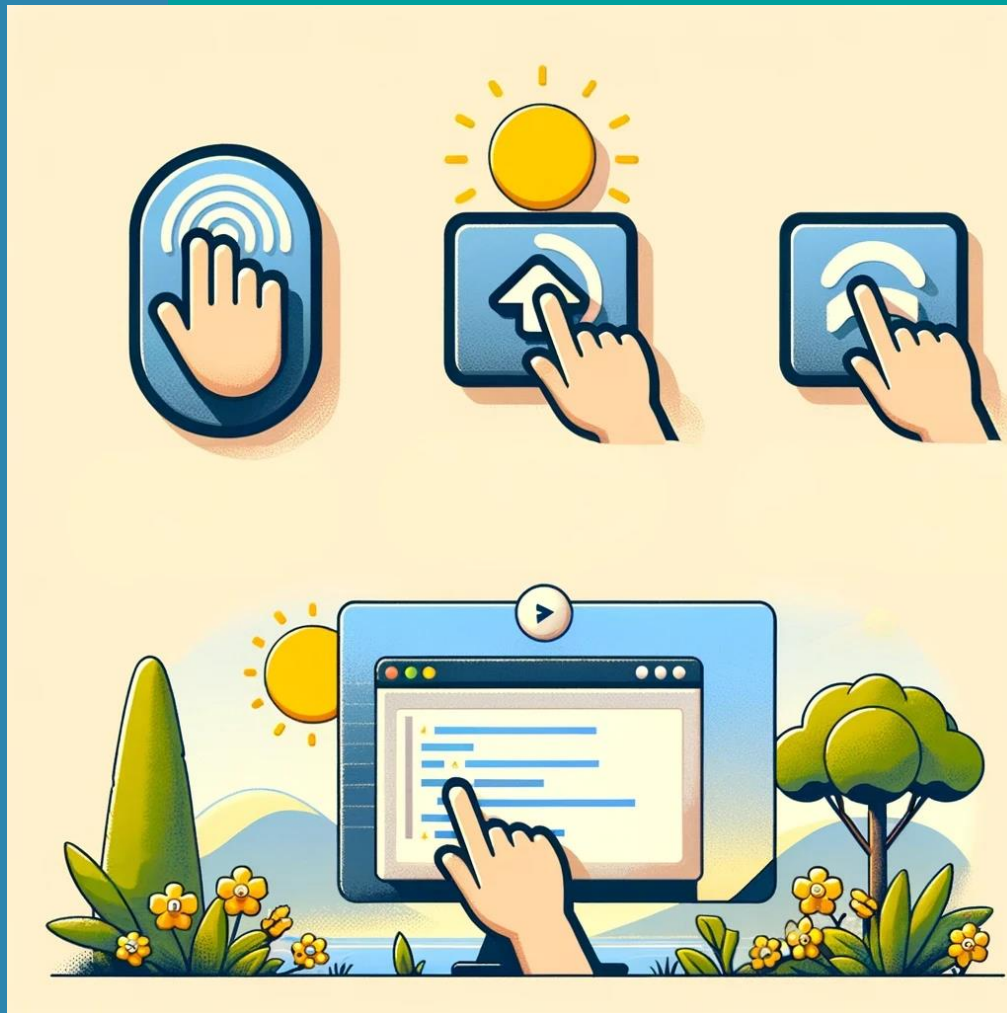


[https://www.linkedin.com/in/
michał-malinowski-malkon/](https://www.linkedin.com/in/michał-malinowski-malkon/)

Wprowadzenie do jQuery

Wykład 05

JĘZYKI PROGRAMOWANIA
STUDIUM PRZYPADKU



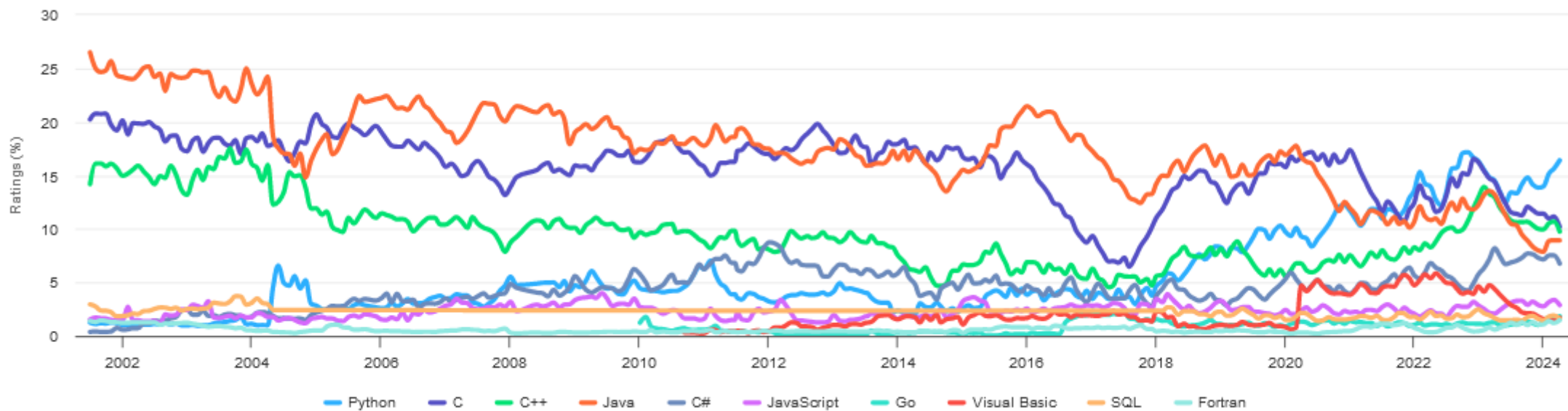
Zagadnienia (2h)

- [Rankingi](#)
- [Informacje podstawowe](#)
- [Dodanie biblioteki do kodu](#)
- [Selektory](#)
- [Manipulacja DOM](#)
- [Efekty i animacje](#)
- [Ajax](#)
- [Łącuchowanie](#)
- [Obsługa błędów](#)
- [Metody pomocnicze](#)

Ranking - Języki programowania

Apr 2024	Apr 2023	Change	Programming Language		Ratings	Change
1	1			Python	16.41%	+1.90%
2	2			C	10.21%	-4.20%
3	4	▲		C++	9.76%	-3.20%
4	3	▼		Java	8.94%	-4.29%
5	5			C#	6.77%	-1.44%
6	7	▲		JavaScript	2.89%	+0.79%
7	10	▲		Go	1.85%	+0.57%
8	6	▼		Visual Basic	1.70%	-2.70%
9	8	▼		SQL	1.61%	-0.06%
10	20	▲▲		Fortran	1.47%	+0.88%
11	11			Delphi/Object Pascal	1.47%	+0.24%
12	12			Assembly language	1.30%	+0.26%
13	18	▲▲		Ruby	1.24%	+0.58%
14	17	▲		Swift	1.23%	+0.51%
15	15			Scratch	1.14%	+0.35%
16	14	▼		MATLAB	1.11%	+0.25%
17	9	▼▼		PHP	1.09%	-0.26%
18	38	▲▲		Kotlin	1.05%	+0.80%
19	19			Rust	1.03%	+0.41%
20	16	▼▼		R	0.84%	+0.09%

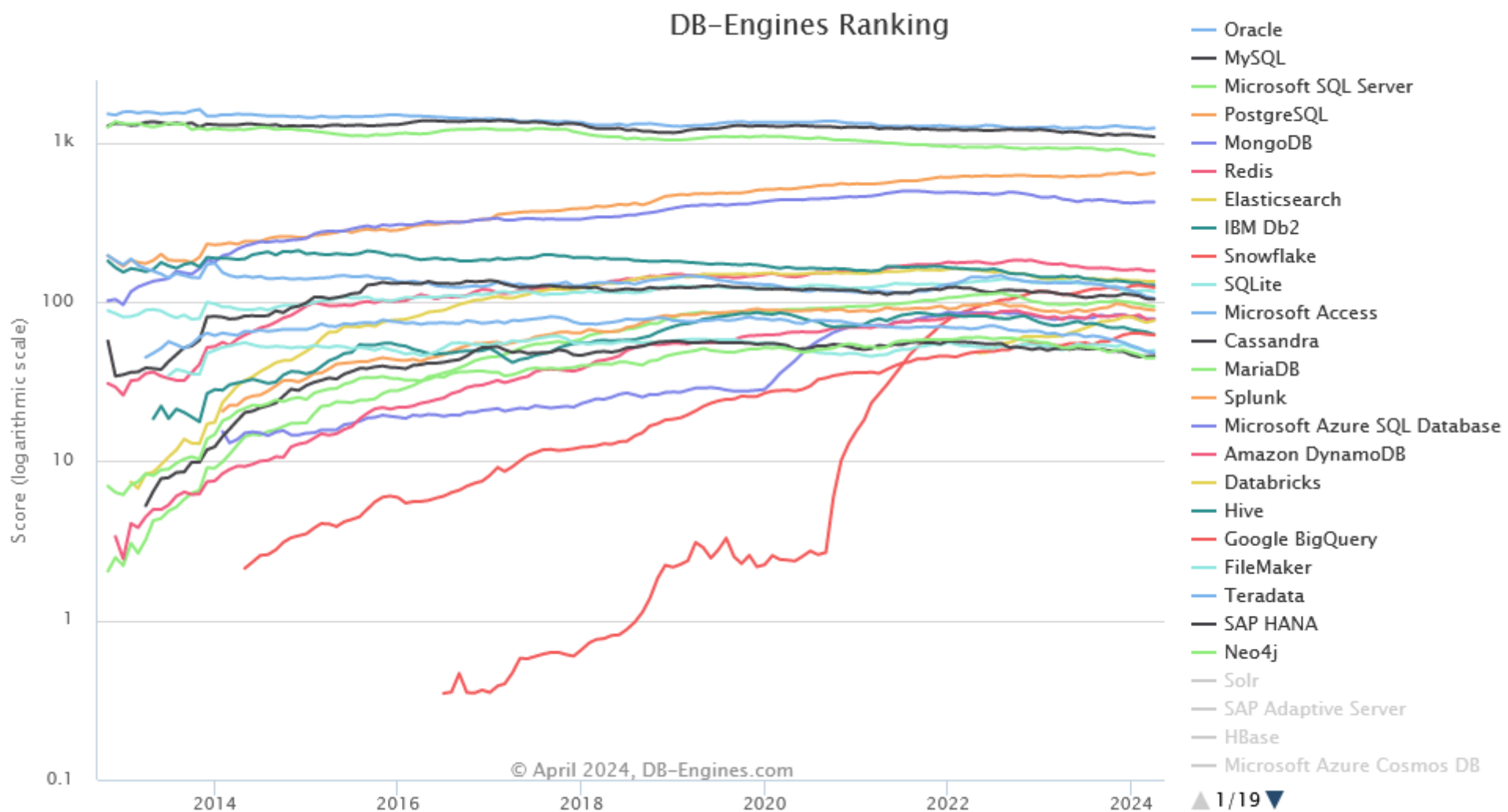
Ranking - Języki programowania



Ranking - Systemy baz danych

Rank			DBMS	Database Model	Score		
Apr 2024	Mar 2024	Apr 2023			Apr 2024	Mar 2024	Apr 2023
1.	1.	1.	Oracle	Relational, Multi-model	1234.27	+13.21	+5.99
2.	2.	2.	MySQL	Relational, Multi-model	1087.72	-13.77	-70.06
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	829.80	-16.01	-88.73
4.	4.	4.	PostgreSQL	Relational, Multi-model	645.05	+10.15	+36.64
5.	5.	5.	MongoDB	Document, Multi-model	423.96	-0.57	-17.93
6.	6.	6.	Redis	Key-value, Multi-model	156.44	-0.56	-17.11
7.	7.	8.	Elasticsearch	Search engine, Multi-model	134.78	-0.01	-6.29
8.	8.	7.	IBM Db2	Relational, Multi-model	127.49	-0.26	-18.00
9.	9.	12.	Snowflake	Relational	123.20	-2.18	+12.07
10.	10.	9.	SQLite	Relational	116.01	-2.15	-18.53
11.	11.	10.	Microsoft Access	Relational	105.40	-2.52	-25.97
12.	12.	11.	Cassandra	Wide column, Multi-model	103.86	-0.72	-7.94
13.	13.	13.	MariaDB	Relational, Multi-model	93.81	-1.22	-2.13
14.	14.	14.	Splunk	Search engine	88.71	-0.97	+3.27
15.	15.	15.	Microsoft Azure SQL Database	Relational, Multi-model	78.40	-0.11	-0.66
16.	16.	16.	Amazon DynamoDB	Multi-model	77.57	-0.15	+0.12
17.	17.	19.	Databricks	Multi-model	76.33	+1.99	+15.36
18.	18.	17.	Hive	Relational	62.59	-2.24	-9.07
19.	19.	20.	Google BigQuery	Relational	61.90	-0.77	+8.59
20.	21.	23.	FileMaker	Relational	49.73	+0.92	-0.27
21.	20.	18.	Teradata	Relational, Multi-model	47.84	-1.10	-13.75
22.	22.	22.	SAP HANA	Relational, Multi-model	45.84	+0.35	-5.24
23.	23.	21.	Neo4j	Graph	44.47	+0.11	-7.13
24.	24.	24.	Solr	Search engine, Multi-model	44.28	+0.37	-3.94
25.	25.	25.	SAP Adaptive Server	Relational, Multi-model	37.12	-0.37	-5.75

Ranking - Systemy baz danych



Informacje podstawowe

jQuery (JavaScript Queries)

to biblioteka JavaScript, która upraszcza manipulację DOM, obsługę zdarzeń, animacje oraz komunikację Ajax, umożliwiając szybsze i łatwiejsze tworzenie interaktywnych stron internetowych.

Pozwala na dynamiczne zarządzanie kontentem i zachowaniami na stronie.



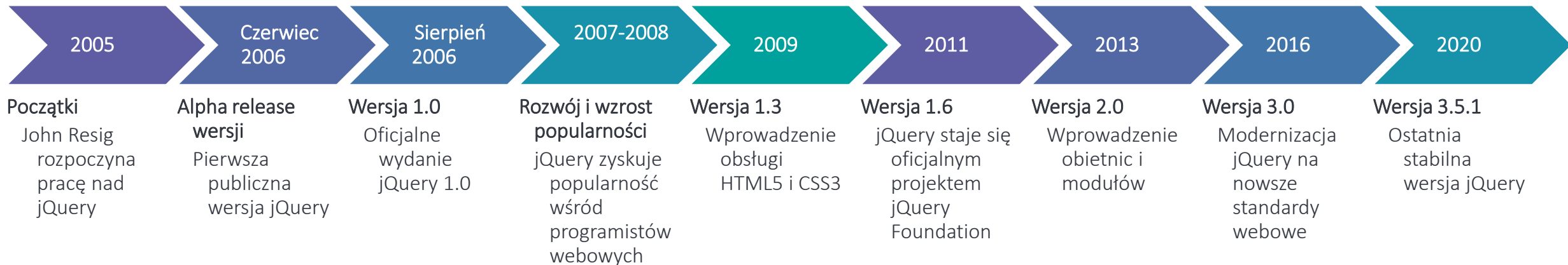
Ajax (Asynchronous JavaScript and XML)

to technika programowania stosowana do tworzenia interaktywnych aplikacji internetowych. Pozwala na asynchroniczne aktualizacje danych na stronie, co oznacza, że można wysyłać i odbierać dane z serwera w tle, bez konieczności przeładowywania całej strony.



Informacje podstawowe

Historia



Informacje podstawowe

Plusy jQuery

Łatwość Użycia

Uproszcza skryptowanie JavaScript, umożliwiając szybkie wybieranie elementów DOM, aplikowanie efektów, manipulowanie zawartością i więcej.

Łańcuchowanie Metod

Umożliwia wykonywanie wielu operacji na elementach za pomocą jednego ciągu wywołań metod, co prowadzi do bardziej zwięzłego kodu.

Manipulacja DOM

Oferuje potężne narzędzia do łatwego dodawania, usuwania i modyfikowania elementów oraz ich atrybutów.

Obsługa Zdarzeń

Zapewnia prosty mechanizm do obsługi zdarzeń, który działa konsekwentnie w różnych przeglądarkach.

Efekty i Animacje

Zawiera bogaty zestaw wbudowanych efektów i animacji, które można łatwo zaimplementować do elementów strony.

Ajax

Prosty, ale potężny interfejs Ajax, umożliwiający ładowanie danych asynchronicznie bez potrzeby przeładowywania całej strony.

Wtyczki

Obszerna biblioteka wtyczek stworzonych przez społeczność, która rozszerza funkcjonalność jQuery.

Wsparcie Wielu Przeglądarek

Automatyczne radzenie sobie z różnicami w implementacji JavaScript i DOM w różnych przeglądarkach.

Mały Rozmiar

Lekkie, szybkie w ładowaniu, nie obciąża nadmiernie wydajności strony.

Informacje podstawowe

Minusy jQuery

Wydajność

jQuery jest wolniejsze niż czysty JavaScript (zwany również Vanilla JS) w niektórych operacjach, szczególnie przy intensywnym manipulowaniu DOM i przetwarzaniu dużych ilości danych.

Nadmiarowość

Współczesne przeglądarki znacznie poprawiły wsparcie dla czystego JavaScript, co czyni wiele funkcji jQuery zbędnymi. Metody takie jak selektory, obsługa zdarzeń i AJAX są teraz natywnie dobrze obsługiwane przez JavaScript.

Wielkość biblioteki

Chociaż jQuery jest stosunkowo lekkie, dodawanie całej biblioteki do projektu dla kilku funkcjonalności może być nieefektywne, szczególnie w projektach, gdzie każdy bajt ma znaczenie dla wydajności strony.

Zależność od zewnętrznej biblioteki

Używanie jQuery jako zewnętrznej zależności może prowadzić do problemów z zależnościami i utrudniać utrzymanie kodu, szczególnie gdy projekt zależy od specyficznych wersji biblioteki.

Mniej odpowiednie dla nowoczesnych aplikacji jednostronicowych (SPA)

Biblioteki takie jak React, Angular, czy Vue.js oferują bardziej kompleksowe podejście do budowy aplikacji SPA i są lepiej przystosowane do zarządzania stanem i interfejsem użytkownika w skomplikowanych aplikacjach.

Zmniejszające się wsparcie i rozwój

W miarę jak nowsze technologie zyskują na popularności, społeczność deweloperów i wsparcie dla jQuery mogą maleć, co może wpływać na ilość dostępnych aktualizacji i wtyczek.

Informacje podstawowe

Dołączenie biblioteki jQuery
załadowanie biblioteki jQuery
z CDN (Content Delivery
Network) Google.

Metoda `$(document).ready()`
używana do wykonania funkcji
JavaScript dopiero po
załadowaniu całego drzewa
DOM strony

Selektor `$("p")` i metoda `.slideDown()`
wybiera element `<p>` i automatycznie
przy ładowaniu strony aplikuje efekt
`slideDown()`, który rozwija element
pionowo, tworząc animację
rozwijania.
Opcja `'slow'` określa, że animacja
powinna być wykonana wolno

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Przykład jQuery</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>
  <h1>Witaj w świecie jQuery!</h1>
  <p style="display: none;">Jestem paragrafem ukrytym przed załadowaniem.</p>

  <script>
    $(document).ready(function() {
      // Efekt fadeIn dla elementu h1 po kliknięciu
      $("h1").click(function() {
        $(this).fadeIn('slow');
      });

      // Efekt slideDown dla elementu p przy ładowaniu strony
      $("p").slideDown('slow');
    });
  </script>
</body>
</html>
```

Stylowanie CSS w elemencie `<p>`
element `<p>` ma zastosowany styl
CSS `display: none;`, co oznacza, że
jest on ukryty na stronie do czasu
wywołania przez jQuery efektu
`slideDown()`

Selektor `$("h1")` i metoda `.click()`
wybiera element HTML `<h1>` i przypisuje
mu funkcję, która będzie wywoływana
przy każdym kliknięciu na ten element

Metody `.fadeIn()`
do płynnego pokazania elementu z efektem
przezroczystości.

Dodanie biblioteki do kodu

Metody dodania

1. Dołączanie jQuery z CDN (Content Delivery Network):
 - Google CDN
 - Microsoft CDN
 - CDNJS Cloudflare
2. Lokalne hostowanie jQuery:
 - Pobierz minifikowaną wersję jQuery z oficjalnej strony jQuery.
 - Umieść plik w katalogu swojego projektu.
 - Dołącz plik w swoim pliku HTML
- Dołączanie jQuery w środowisku Node.js:
 - Instalacja jQuery za pomocą npm (npm install jquery)
 - Instalacja jQuery za pomocą yarn (yarn add jquery)
 - Importowanie jQuery w skryptach JavaScript używając require lub import

```
<!-- Dołączanie najnowszej wersji jQuery z Google CDN -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<!-- Dołączanie jQuery z Microsoft CDN -->
<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.5.1.min.js"></script>

<!-- Dołączanie jQuery z CDNJS Cloudflare -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

```
<script src="ścieżka/do/pliku/jquery-3.5.1.min.js"></script>
```

Dodanie biblioteki do kodu

Oficjalna strona jQuery

<https://jquery.com/download/>

Download jQuery

Latest version

To locally download these files, right-click the link and select "Save as..." from the menu.

Download the compressed, production version:

[Download jQuery 3.7.1](#)

- [Download the uncompressed development version of jQuery 3.7.1](#)
- [Download the map file for jQuery 3.7.1](#)
- [jQuery 3.7.1 blog post with release notes](#)

The slim build is a smaller version, that excludes the [ajax](#) and [effects](#) modules:

- [Download jQuery 3.7.1 slim build](#)
- [Download the uncompressed development version of the jQuery 3.7.1 slim build](#)
- [Download the map for the jQuery 3.7.1 slim build](#)

Selektory

Selektory

pozwalają na szybkie i efektywne przeszukiwanie drzewa DOM w celu znalezienia konkretnych elementów, z którymi chcemy pracować.

Można je wykorzystać do zmiany stylów, zawartości, obsługi zdarzeń i wiele innych.

Przykłady Selektorów

`$('div')` - wybiera wszystkie elementy po tagu.

`$('.klasa')` - wybiera wszystkie elementy o danej klasie.

`$('#id')` - wybiera element o określonym ID.

`$('ul li:first-child')` - wybiera pierwsze dzieci elementów `<li>` w każdym `<ul>`.

```
<div>Div 1</div>
<div>Div 2</div>
<div class="klasa">Element z klasą 'klasa'</div>
<div id="unikalneID">Element z unikalnym ID</div>
<ul><li>Pierwszy element listy</li><li>Drugi element listy</li></ul>
<ul><li>Kolejny pierwszy element</li><li>Kolejny drugi element</li></ul>

<script>
  $(document).ready(function() {
    $('div').css('border', '1px solid blue');
    $('.klasa').css('font-weight', 'bold');
    $('#unikalneID').css('text-decoration', 'underline');
    $('ul li:first-child').css('background-color', 'yellow');
  });
</script>
```

Selektory

Selektory podstawowe

Uniwersalny (*): Wybiera wszystkie elementy na stronie.

Typu (tagname): Wybiera wszystkie elementy danego typu, np. \$('div') dla wszystkich elementów <div>.

ID (#id): Wybiera pojedynczy element o danym ID, np. \$('#mojeId').

Klasy (.klasa): Wybiera wszystkie elementy, które mają określoną klasę, np. \$('.mojaKlasa').

```
<div>Div 1</div>
<div class="mojaKlasa">Div z klasą</div>
<div id="mojeId">Div z ID</div>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    $(document).ready(function() {
        $('*').css('border', '1px solid #ddd'); // Uniwersalny selektor
        $('div').css('background', '#efefef'); // Selektor typu
        $('#mojeId').css('color', 'blue'); // Selektor ID
        $('.mojaKlasa').css('font-size', '20px'); // Selektor klasy
    });
</script>
```

Selektory

Selektory hierarchii

Potomka (parent descendant): Wybiera wszystkie elementy potomne danego rodzica, np. `$('div span')` dla wszystkich `<span>` w `<div>`.

Dziecka (parent > child): Wybiera bezpośrednie dzieci elementu, np. `$('div > p')` dla bezpośrednich dzieci `<p>` w `<div>`.

Następnika (prev + next): Wybiera bezpośredniego następcę elementu, np. `$('div + p')` dla `<p>` bezpośrednio po `<div>`.

Wszechobecne rodzeństwo (prev ~ siblings): Wybiera wszystkie elementy rodzeństwa na tym samym poziomie po danym elemencie, np. `$('div ~ p')`.

```
<div class="parent">
  <span>Span wewnątrz div (potomek)</span>
  <p>Paragraf wewnątrz div (dziecko)</p>
  <div><span>Span wewnątrz drugiego div (potomek)</span></div>
</div>
<p>Paragraf poza div (następnik)</p>
<div class="sibling"><p>Paragraf wewnątrz kolejnego div (rodzeństwo)</p></div>
<p>Paragraf poza div (rodzeństwo)</p>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(document).ready(function() {
  $('.parent span').addClass('highlight');
  $('.parent > p').css('font-weight', 'bold');
  $('.parent + p').css('color', 'blue');
  $('.sibling ~ p').css('text-decoration', 'underline');
});
</script>
```

Selektory

Selektory atrybutów

Atrybutu ([attribute]): Wybiera elementy, które mają określony atrybut, np. `$('input[type]')` dla wszystkich `<input>` z atrybutem `type`.

Atrybutu z wartością ([attribute="value"]): Wybiera elementy, które mają atrybut o określonej wartości, np. `$('input[type="checkbox"]')`.

Atrybutu zawierającego wartość ([attribute*="value"]): Wybiera elementy, które mają atrybut zawierający podaną wartość w dowolnym miejscu np. `$('input[type*="example"]')`.

```
<input type="text" name="email" value="user@example.com" />
<input type="checkbox" name="subscribe" checked />
<input type="text" name="email2" value="admin@example.com" />

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(document).ready(function() {
    $('input[type]').addClass('highlight'); // Wszystkie <input> z atrybutem type
    $('input[type="checkbox"]').css('margin', '10px'); // Wszystkie <input> type="checkbox"
    $('input[value*="example"]').css('border', '1px solid red'); // Wszystkie <input> z wartością zawierającą "example"
});
</script>
```

Selektory

Selektory formularzy

Wprowadzenie (:input): Wybiera wszystkie elementy formularza.

Tekstowe (:text): Wybiera wszystkie pola tekstowe.

Wybrane (:selected): Wybiera elementy, które są zaznaczone

```
<form id="myForm">
  <input type="text" name="first_name" value="Jan">
  <input type="text" name="last_name" value="Kowalski">
  <select name="city">
    <option value="Warsaw">Warszawa</option>
    <option value="Cracow" selected>Kraków</option>
  </select>
</form>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
  $(document).ready(function() {
    $(':input').css('border', '2px solid blue');
    $(':text').css('background-color', 'lightgrey');
    $(':selected').css('font-weight', 'bold');
  });
</script>
```

Selektory

Pseudoklasy

Pierwszego dziecka (:first-child): Wybiera pierwsze dziecko swojego rodzica.

Ostatniego dziecka (:last-child): Wybiera ostatnie dziecko swojego rodzica.

N-tego dziecka (:nth-child(n)): Wybiera n-te dziecko swojego rodzica.

Pierwszego typu (:first-of-type): Wybiera pierwszy element danego typu w swoim rodzicu.

Jedynego dziecka (:only-child): Wybiera elementy, które są jedynymi dziećmi swoich rodziców.

```
<ul id="myList">
  <li>Jabłko</li>
  <li>Pomarańcza</li>
  <li>Banany</li>
  <li>Gruszka</li>
</ul>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
  $(document).ready(function() {
    $('#myList li:first-child').addClass('highlight');
    $('#myList li:last-child').addClass('highlight');
    $('#myList li:nth-child(2)').addClass('highlight');
    $('#myList li:first-of-type').addClass('highlight');
    $('#myList li:only-child').addClass('highlight');
  });
</script>
```

Manipulacja DOM

Dodawanie nowej zawartości

.append(): Dodaje zawartość na końcu wybranych elementów.

.prepend(): Wstawia zawartość na początku wybranych elementów.

.after(): Wstawia zawartość bezpośrednio po wybranych elementach.

.before(): Wstawia zawartość bezpośrednio przed wybranymi elementami.

```
<div id="container">
  <h2>Witaj na stronie!</h2>
</div>

<script>
$(document).ready(function() {
  $('#container').append('<p>Dodany na końcu kontenera.</p>');
  $('#container').prepend('<p>Dodany na początku kontenera.</p>');
  $('#container').after('<div>Zawartość po kontenerze.</div>');
  $('#container').before('<div>Zawartość przed kontenerem.</div>');
});
</script>
```

Manipulacja DOM

Modyfikacja zawartości

.html(): Zmienia zawartość HTML wewnątrz wybranych elementów.

.text(): Zmienia tekstową zawartość wybranych elementów, usuwając wszelki HTML.

Modyfikacja struktury dokumentu

.remove(): Usuwa wybrane elementy (wraz z ich potomkami) z dokumentu.

.empty(): Usuwa wszystkie potomne elementy wewnątrz wybranych elementów, pozostawiając same puste kontenery

```
<div id="content">Oryginalna zawartość</div>
<button id="changeContent">Zmień zawartość</button>
<button id="changeText">Zmień tekst</button>
<button id="removeElement">Usuń element</button>
<button id="emptyElement">Opróżnij element</button>

<script>
$(document).ready(function() {
    $('#changeContent').click(function() {
        $('#content').html('<p>Zmieniona zawartość HTML</p>');
    });
    $('#changeText').click(function() {
        $('#content').text('Zmieniony tekst, bez HTML');
    });
    $('#removeElement').click(function() {
        $('#content').remove();
    });
    $('#emptyElement').click(function() {
        $('#content').empty();
    });
});
</script>
```

Efekty i animacje

Reakcja na interakcje użytkownika

.show(): Natychmiastowo pokazuje ukryty element, sprawiając, że staje się widoczny.

.hide(): Natychmiastowo ukrywa element, czyniąc go niewidocznym dla użytkownika.

.toggle(): Przełącza stan widoczności elementu — jeśli jest ukryty, pokazuje go; jeśli widoczny, ukrywa.

.fadeIn(): Stopniowo zwiększa przezroczystość ukrytego elementu, aż stanie się on w pełni widoczny, tworząc efekt pojawiania się.

.fadeOut(): Stopniowo zmniejsza przezroczystość elementu do całkowitego zniknięcia, tworząc efekt znikania.

.slideDown(): Stopniowo "rozsuwa" element, odsłaniając go i tworząc efekt rozwijania się z góry do dołu.

.slideUp(): Stopniowo "zsuwa" element, ukrywając go i tworząc efekt zwijania się z dołu do góry

```
<html lang="pl">
<head>
<meta charset="UTF-8">
<title>Przykłady jQuery</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<style>
.box { width: 100px; height: 100px; background-color: #007bff; margin: 10px 0; display: none; }
</style>
</head>
<body>

<div id="box" class="box"></div>

<button onclick="$('#box').show()">Show</button>
<button onclick="$('#box').hide()">Hide</button>
<button onclick="$('#box').toggle()">Toggle</button>
<button onclick="$('#box').fadeIn()">Fade In</button>
<button onclick="$('#box').fadeOut()">Fade Out</button>
<button onclick="$('#box').slideDown()">Slide Down</button>
<button onclick="$('#box').slideUp()">Slide Up</button>

</body>
</html>
```

Ajax

Ajax (Asynchronous JavaScript and XML)

to technika umożliwiająca asynchroniczne przesyłanie danych między klientem a serwerem, co pozwala na aktualizację części strony internetowej bez konieczności przeładowywania całej strony.

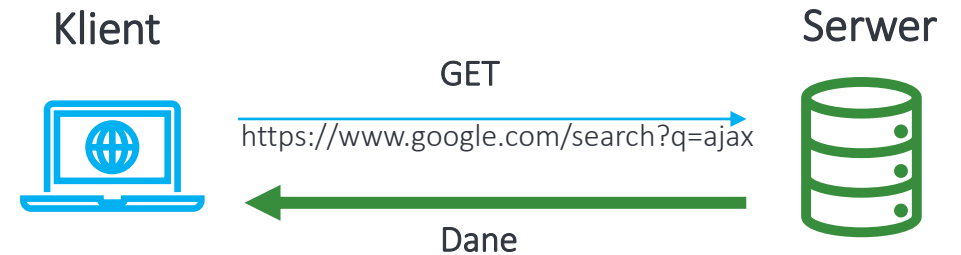
Metoda	Typ Żądania	Prostota	Kontrola	Zastosowanie typowe
.ajax()	Dowolny	Niska	Pełna	Skomplikowane zapytania, konfiguracja nagłówków, obsługa różnych formatów danych, aplikacje SPA, zaawansowane interakcje z REST API.
.get()	GET	Wysoka	Ograniczona	Proste pobieranie danych (tekst, JSON), szybkie aktualizacje bez konieczności pełnej konfiguracji.
.post()	POST	Wysoka	Ograniczona	Wysyłanie danych, takich jak formularze i informacje użytkownika, zwykle przy przesyłaniu danych do serwera.
.load()	GET	Średnia	Ograniczona	Dynamiczne ładowanie treści HTML do określonej części strony, aktualizacje sekcji bez przeładowywania całej strony.

Ajax

GET

służy do pobierania danych z serwera.

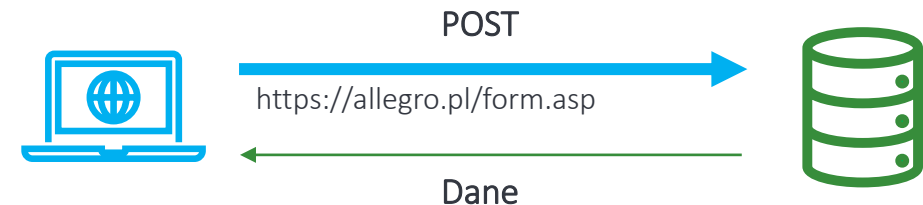
Dane te są zazwyczaj zwracane w formacie HTML, JSON lub XML i mogą być wyświetlane użytkownikowi w przeglądarce internetowej.



POST

służy do wysyłania danych do serwera w celu wykonania określonej akcji.

Dane te mogą być formularzami, plikami lub dowolnymi innymi danymi, które mają zostać przetworzone przez serwer.



Ajax

GET vs POST

Aspekt	GET	POST
Wizualizacja danych	Parametry są dołączane do URL, co sprawia, że są widoczne dla użytkownika.	Dane są wysyłane w treści żądania, zwykle nie są widoczne dla użytkownika.
Ograniczenia długości	Ograniczona długość URL przez przeglądarkę, co wpływa na ilość danych.	Teoretycznie brak ograniczeń długości, może przysyłać dużo większe ilości danych.
Zapisywanie w cache	Możliwość zapisywania w pamięci cache.	Zwykle odpowiedzi nie są cachowane.
Bezpieczeństwo	Mniej bezpieczny, ponieważ dane są wyświetlane w URL.	Bezpieczniejszy dla poufnych danych, ponieważ nie są wyświetlane w URL.
Historia przeglądarki	Parametry pozostają w historii przeglądarki.	Dane nie pozostają w historii przeglądarki.
Typy danych	Ograniczone głównie do tekstowych danych zapytania.	Może przysyłać różnorodne typy danych, w tym binarne.
Zastosowanie	Pobieranie danych, np. wyszukiwanie, odświeżanie treści. Operacje, które mogą być powtarzane bez skutków ubocznych. Żądania, które mogą być zakodowane w linku (np. zakładki).	Wysyłanie danych do serwera, np. przysyłanie formularzy. Operacje, które powinny być wykonywane tylko raz, np. złożenie zamówienia. <u>Przesyłanie dużych lub poufnych danych.</u>

Ajax

Metoda \$.ajax()

Uniwersalna metoda do tworzenia żądań Ajax, oferująca pełną kontrolę nad konfiguracją żądania.

```
$.ajax({  
  url: 'ścieżka/do/zasobu',    // URL żądania  
  type: 'GET',                // Metoda HTTP, 'GET' lub 'POST'  
  dataType: 'json',           // Oczekiwany typ danych odpowiedzi, np. 'json', 'xml', 'html', itp.  
  data: {                     // Dane wysyłane z żądaniem  
    klucz1: 'wartość1',  
    klucz2: 'wartość2'  
  },  
  success: function(data) {    // Funkcja wywoływana w przypadku sukcesu  
    // Obsługa danych 'data' zwróconych z serwera  
  },  
  error: function(jqXHR, textStatus, errorThrown) { // Funkcja wywoływana w przypadku błędu  
    // Obsługa błędu  
  }  
});
```

```
<html lang="pl">  
<head>  
  <meta charset="UTF-8">  
  <title>AJAX w jQuery</title>  
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>  
  <script>  
    $(function() {  
      $('#ajaxRequestButton').on('click', function() {  
        $.ajax({  
          url: 'ścieżka/do/zasobu',  
          type: 'GET',  
          dataType: 'json',  
          success: function(data) {  
            $('#responseContainer').text(JSON.stringify(data));  
          },  
          error: function() {  
            $('#responseContainer').text('Błąd żądania');  
          }  
        });  
      });  
    });  
  </script>  
</head>  
<body>  
  
  <button id="ajaxRequestButton">Wyślij żądanie AJAX</button>  
  <div id="responseContainer"></div>  
  
</body>  
</html>
```

Ajax

Metoda \$.get()

Metoda do wysyłania żądań GET.

```
$.get('ścieżka/do/zasobu',    // URL żądania
    {                        // Dane wysyłane z żądaniem
        klucz1: 'wartość1',
        klucz2: 'wartość2'
    },
    function(data) {          // Funkcja wywoływana w przypadku sukcesu
        // Obsługa danych 'data' zwróconych z serwera
    },
    'json'                    // Oczekiwany typ danych odpowiedzi, np. 'json'
).fail(function(jqXHR, textStatus, errorThrown) { // Funkcja wywoływana w przypadku błędu
    // Obsługa błędu
});
```

```
<html lang="pl">
<head>
<meta charset="UTF-8">
<title>Przykład użycia jQuery .get()</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(document).ready(function(){
    $('#ajaxRequestButton').click(function(){
        $.get('ścieżka/do/zasobu', { klucz1: 'wartość1', klucz2: 'wartość2' }, function(data) {
            $('#responseContainer').text(JSON.stringify(data));
        }, 'json')
        .fail(function(jqXHR, textStatus, errorThrown) {
            $('#responseContainer').text('Wystąpił błąd: ' + textStatus);
        });
    });
});
</script>
</head>
<body>

<button id="ajaxRequestButton">Wyślij żądanie GET</button>
<div id="responseContainer">Odpowiedź serwera zostanie wyświetlona tutaj.</div>

</body>
</html>
```

Ajax

Metoda \$.load()

Służy do ładowania danych z określonego URL-a i wstawiania ich bezpośrednio do wybranego elementu DOM.

```
$('#elementId').load('ścieżka/do/zasobu',    // URL żądania
{                                              // Dane wysyłane z żądaniem
  klucz1: 'wartość1',
  klucz2: 'wartość2'
},
function(responseText, textStatus, xhr) {    // Funkcja wywoływana po zakończeniu żądania
  if (textStatus == "error") {
    // Obsługa błędu, np. wyświetlenie komunikatu
  } else {
    // Obsługa danych 'responseText' zwróconych z serwera
  }
});
```

```
<html lang="pl">
<head>
<meta charset="UTF-8">
<title>Przykład jQuery .load()</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(document).ready(function(){
  // Przykładowe wywołanie może być zainicjowane przez zdarzenie, np. kliknięcie przycisku.
  $('#loadDataButton').click(function() {
    $('#elementId').load('ścieżka/do/zasobu', {
      klucz1: 'wartość1',
      klucz2: 'wartość2'
    }, function(responseText, textStatus, xhr) {
      if (textStatus == "error") {
        $('#elementId').text('Wystąpił błąd: ' + xhr.status + " " + xhr.statusText);
      } else {
        // Tutaj 'responseText' zawiera treść załadowaną z 'ścieżka/do/zasobu'
        // Jeśli jest potrzebna dodatkowa logika, można ją dodać tutaj.
      }
    });
  });
});
</script>
</head>
<body>

<button id="loadDataButton">Załaduj dane</button>
<div id="elementId">Tu pojawią się załadowane dane.</div>

</body>
</html>
```

Ajax

Metoda \$.post()

Metoda do wysyłania żądań POST.

```
$.post('ścieżka/do/zasobu', // URL żądania
{
    // Dane wysyłane z żądaniem
    klucz1: 'wartość1',
    klucz2: 'wartość2'
},
function(data) { // Funkcja wywoływana w przypadku sukcesu
    // Obsługa danych 'data' zwróconych z serwera
},
'json' // Oczekiwany typ danych odpowiedzi, np. 'json'
).fail(function(jqXHR, textStatus, errorThrown) { // Funkcja wywoływana w przypadku błędu
    // Obsługa błędu
});
```

```
<html lang="pl">
<head>
<meta charset="UTF-8">
<title>Przykład z jQuery .post()</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(document).ready(function(){
    $('#ajaxRequestButton').click(function(){
        $.post('ścieżka/do/zasobu', { klucz1: 'wartość1', klucz2: 'wartość2' }, function(data) {
            $('#responseContainer').text(JSON.stringify(data));
        }, 'json')
        .fail(function(jqXHR, textStatus, errorThrown) {
            $('#responseContainer').text('Wystąpił błąd: ' + textStatus);
        });
    });
});
</script>
</head>
<body>

<button id="ajaxRequestButton">Wyślij żądanie POST</button>
<div id="responseContainer">Odpowiedź serwera zostanie wyświetlona tutaj.</div>

</body>
</html>
```

Ajax

Synchroniczność

oznacza wykonywanie operacji sekwencyjnie, jedna po drugiej.

Oznacza to, że program musi zakończyć jedną operację, zanim rozpocznie inną.



Asynchroniczność

oznacza wykonywanie operacji niezależnie od siebie.

Oznacza to, że program nie musi czekać na zakończenie jednej operacji, aby rozpocząć inną.



Ajax

Synchroniczność vs Asynchroniczność

Aspekt	Synchroniczność	Asynchroniczność
Zastosowanie	Sekwencyjne operacje; prostsze zadania i skrypty.	Operacje I/O; interaktywne aplikacje; zadania w tle.
Zalety	Prosta logika przepływu; łatwa obsługa błędów.	Nie blokuje interfejsu użytkownika; wydajność; skalowalność.
Wady	Może blokować UI; nieefektywne dla operacji I/O.	Trudniejsza obsługa błędów; złożoność w zarządzaniu stanem.
Responsywność UI	Niska podczas długotrwałych operacji.	Wysoka, ponieważ UI jest dostępny podczas przetwarzania.
Wydajność	Może być niska dla wielu jednoczesnych operacji I/O.	Wysoka, pozwala na jednoczesne przetwarzanie wielu zadań.
Skalowalność	Ograniczona w zależności od czasu trwania operacji.	Wysoka, umożliwia obsługę wielu żądań jednocześnie.
Obsługa błędów	Bezpośrednia; sekwencyjna w miejscu wystąpienia.	Może wymagać złożonych mechanizmów.
Przepływ kontroli	Linearny i przewidywalny.	Może być nielinearny i wymagać szczególnego zarządzania.

UI – interfejs użytkownika

operacji I/O (InPut/OutPut) – operacje wejścia/wyjścia

Ajax

Synchroniczność w jQuery

jest obecnie rzadko używane i generalnie uznawane za złą praktykę ze względu na negatywny wpływ na UI.

Ustawienie async: false prowadzi do synchronicznego żądania

```
$.ajax({  
  url: 'ścieżka/do/zasobu',  
  async: false, // Niezalecane, używaj z ostrożnością!  
  success: function(data) {  
    // obsługa danych  
  }  
});
```

Asynchroniczność w jQuery

jest standardem w jQuery i w programowaniu webowym.

Ustawienie async: true prowadzi do asynchronicznego żądania i jest to wartość domyślna

```
$.ajax({  
  url: 'ścieżka/do/zasobu',  
  success: function(data) {  
    // obsługa danych  
  }  
});
```

Dotyczy to metod \$.ajax, \$.get, \$.post, \$.load

Łańcuchowanie

Łańcuchowanie

Łańcuchowanie to potężna cecha jQuery, która pozwala wykonywać wiele operacji na elementach w jednej linii kodu, zwiększając czytelność i efektywność.

Korzyści: Uproszczenie kodu, mniejsza potrzeba tworzenia tymczasowych zmiennych, szybsze wykonanie.

Zasada działania: Każda metoda jQuery zwraca obiekt jQuery, umożliwiając wywołanie kolejnej metody.

Dobre praktyki

Używaj łańcuchowania, aby zmniejszyć ilość kodu i poprawić jego czytelność.

Zachowaj umiar – zbyt długie łańcuchy mogą stać się mniej zrozumiałe.

Rozważ używanie zmiennych pomocniczych, jeśli łańcuch staje się skomplikowany.

Przykład bez łańcuchowania

```
$('#element').addClass('klasa1');  
$('#element').hide();  
$('#element').slideDown(2000);
```

Przykład z łańcuchowaniem

```
$('#element').addClass('klasa1').hide().slideDown(2000);
```

Bez zmiennej pomocniczej (długi rozmiar)

```
$('#element')  
  .addClass('nowa-klasa')  
  .slideDown('slow')  
  .find('.dziecko')  
    .css('color', 'red')  
  .end()  
  .find('.inny-dziecko')  
    .css('border', '1px solid blue');
```

Ze zmienną pomocniczą

```
var $element = $('#element').addClass('nowa-klasa').slideDown('slow');  
$element.find('.dziecko').css('color', 'red');  
$element.find('.inny-dziecko').css('border', '1px solid blue');
```

Obsługa błędów

Obsługa błędów

Zarządzanie i obsługa błędów jest kluczowa dla stabilności i niezawodności aplikacji webowych

\$.fail(): Obsługa błędów dla żądań AJAX..

\$.ajax({/* opcje */}).fail(function(jqXHR, textStatus, errorThrown) { /* obsługa błędu */ });

\$.done(): Reakcja na pomyślne wykonanie żądania.

\$.ajax({/* opcje */}).done(function(data) { /* logika sukcesu */ });

\$.always(): Logika, która powinna być wykonana niezależnie od tego, czy żądanie się powiodło, czy nie.

\$.ajax({/* opcje */}).always(function() { /* zawsze wykonaj */ });

```
<!DOCTYPE html>
<html lang="pl">
<head>
<meta charset="UTF-8">
<title>Przykład obsługi błędów w jQuery</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(function() {
    $('#ajaxCallButton').on('click', function() {
        $.ajax({ url: 'ścieżka/do/zasobu' })
            .done(function(data) { $('#response').html(data); })
            .fail(function(jqXHR, textStatus) { $('#response').html('Błąd: ' + textStatus); })
            .always(function() { console.log('Żądanie zakończone.'); });
    });
});
</script>
</head>
<body>
<button id="ajaxCallButton">Wykonaj żądanie AJAX</button>
<div id="response"></div>
</body>
</html>
```

Metody pomocnicze

Operacje na kolekcjach

jQuery oferuje szereg metod ułatwiających obsługę typowych zadań związanych z kolekcjami elementów DOM i danymi.

`$.each()`: iteracja po kolekcji elementów jQuery.

`$(selektor).each(function(index, element) { /* logika iteracji */ });`

`$.map()`: transformacja kolekcji elementów w nową tablicę wartości.

`$(selektor).map(function(index, element) { /* logika transformacji */ }).get();`

`$.data()`: Przechowywanie lub odczytywanie danych powiązanych z elementami.

Zapis: `$(selektor).data('klucz', wartość);`

Odczyt: `$(selektor).data('klucz');`

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(function(){
  $('li').each(function(index) {
    console.log(index + ': ' + $(this).text());
  });
  var listTexts = $('li').map(function() {
    return $(this).text();
  }).get();
  console.log(listTexts);
  $('div').data('info', { klucz1: 'wartość1', klucz2: 'wartość2' });
  $('#showDataButton').click(function() {
    alert('Klucz1: ' + $('div').data('info').klucz1 + ', Klucz2: ' + $('div').data('info').klucz2);
  });
});
</script>
</head>
<body>
<ul>
  <li>Element 1</li>
  <li>Element 2</li>
  <li>Element 3</li>
</ul>
<div id="dataDiv"></div>
<button id="showDataButton">Pokaż przechowane dane</button>
</body>
</html>
```

Metody pomocnicze

Operacje na tablicach i obiektach

jQuery dostarcza zbiór funkcji pomocniczych ułatwiających pracę z obiektami i tablicami.

`$.extend()`: Łączenie zawartości dwóch lub więcej obiektów w jeden nowy obiekt.

`$.extend(target, object1, object2, ...)`

`$.merge()`: Scala dwie tablice i zwraca jedną złączoną tablicę.

`$.merge(first, second);`

`$.grep()`: Filtrowanie tablicy za pomocą funkcji testującej.

`$.grep(array, function(element, index) {}, [invert]);`

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
$(function() {
    var obiektA = { a: 1, b: 2 }, obiektB = { b: 3, c: 4 }, scalonyObiekt = $.extend({}, obiektA, obiektB);
    console.log('Scalony obiekt:', scalonyObiekt);
    var pierwszaTablica = [1, 2, 3], drugaTablica = [4, 5, 6], polaczoneTablice = $.merge(pierwszaTablica, drugaTablica);
    console.log('Połączone tablice:', polaczoneTablice);
    var liczby = [0, 1, 2, 3, 4, 5, 6], parzysteLiczby = $.grep(liczby, function(el) { return el % 2 === 0; });
    console.log('Parzyste liczby:', parzysteLiczby);
});
</script>
</head>
<body>
    <h1>Przykłady jQuery - Narzędzia pomocnicze</h1>
    <div>Console.log zawiera wyniki operacji.</div>
</body>
</html>
```

Dodatkowe informacje

Zasoby Internetu

- jQuery
 - <https://jquery.com/>
 - <https://kursjs.pl/kurs/jquery/jquery>
 - <https://www.w3schools.com/jquery/default.asp>
 - https://youtu.be/8zk_A3pD_8c?si=ZLGgksVWGz9sj2bX (film)



